# Babel

**Javier Bezos**
Current maintainer

**Johannes L. Braams**
Original author

Localization and internationalization

Unicode
T<sub>E</sub>X
pdfT<sub>E</sub>X
LuaT<sub>E</sub>X
XeT<sub>E</sub>X

# 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).

LUATEX/XETEX

```
\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.

**NOTE**  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}
```

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

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

**WARNING**  In the preamble the main language has *not* been selected, except hyphenation patterns and the name assigned to \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.

```
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, yi). 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.

`\selectlanguage`    {⟨*language*⟩}

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

```
\selectlanguage{german}
```

This command can be used as environment, too.

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

**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).

9

**\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.
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 convention because it might be enforced in future releases of babel to catch possible errors.

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

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

`\babelshorthand` {⟨*shorthand*⟩}

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

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

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

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

**Languages with no shorthands** Croatian, English (any variety), Indonesian, Hebrew, Interlingua, Irish, Lower Sorbian, Malaysian, North Sami, Romanian, Scottish, Welsh
**Languages with only " as defined shorthand character** Albanian, Bulgarian, Danish, Dutch, Finnish, German (old and new orthography, also Austrian), Icelandic, Italian, Norwegian, Polish, Portuguese (also Brazilian), Russian, Serbian (with Latin script), Slovene, Swedish, Ukrainian, Upper Sorbian
**Basque** " ' ~
**Breton** : ; ? !
**Catalan** " ' `
**Czech** " -
**Esperanto** ^
**Estonian** " ~
**French** (all varieties) : ; ? !
**Galician** " . ' ~ < >
**Greek** ~
**Hungarian** `
**Kurmanji** ^
**Latin** " ^ =
**Slovak** " ^ ' -
**Spanish** " . < > ' ~
**Turkish** : ! =

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

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

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

`\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 უმდიდრესია მთელ მსოფლიოში.

\end{document}
```

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

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

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

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

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

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

Or also:

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

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

```
\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 1ຩ 1ຨ 1ງ 1ກ 1ຯ} % 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] | bem | Bemba |
| agq | Aghem | bez | Bena |
| ak | Akan | bg | Bulgarian[ul] |
| am | Amharic[ul] | bm | Bambara |
| ar | Arabic[ul] | bn | Bangla[ul] |
| ar-DZ | Arabic[ul] | bo | Tibetan[u] |
| ar-MA | Arabic[ul] | brx | Bodo |
| ar-SY | Arabic[ul] | bs-Cyrl | Bosnian |
| as | Assamese | bs-Latn | Bosnian[ul] |
| asa | Asu | bs | Bosnian[ul] |
| ast | Asturian[ul] | ca | Catalan[ul] |
| az-Cyrl | Azerbaijani | ce | Chechen |
| az-Latn | Azerbaijani | cgg | Chiga |
| az | Azerbaijani[ul] | chr | Cherokee |
| bas | Basaa | ckb | Central Kurdish |
| be | Belarusian[ul] | cop | Coptic |

| | | | | |
|---|---|---|---|---|
| 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 | German[ul] | | it | Italian[ul] |
| de | German[ul] | | ja | Japanese |
| 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] | | kn | Kannada[ul] |
| eo | Esperanto[ul] | | ko | Korean |
| es-MX | Spanish[ul] | | kok | Konkani |
| es | Spanish[ul] | | ks | Kashmiri |
| et | Estonian[ul] | | ksb | Shambala |
| eu | Basque[ul] | | ksf | Bafia |
| ewo | Ewondo | | ksh | Colognian |
| fa | Persian[ul] | | kw | Cornish |
| ff | Fulah | | ky | Kyrgyz |
| fi | Finnish[ul] | | lag | Langi |
| fil | Filipino | | lb | Luxembourgish |
| fo | Faroese | | lg | Ganda |
| fr | French[ul] | | lkt | Lakota |
| fr-BE | French[ul] | | ln | Lingala |
| fr-CA | French[ul] | | lo | Lao[ul] |
| fr-CH | French[ul] | | lrc | Northern Luri |
| fr-LU | French[ul] | | lt | Lithuanian[ul] |
| fur | Friulian[ul] | | lu | Luba-Katanga |
| fy | Western Frisian | | luo | Luo |
| ga | Irish[ul] | | luy | Luyia |
| gd | Scottish Gaelic[ul] | | lv | Latvian[ul] |
| gl | Galician[ul] | | mas | Masai |
| grc | Ancient Greek[ul] | | mer | Meru |
| gsw | Swiss German | | mfe | Morisyen |
| gu | Gujarati | | mg | Malagasy |
| guz | Gusii | | mgh | Makhuwa-Meetto |
| gv | Manx | | mgo | Meta' |
| ha-GH | Hausa | | mk | Macedonian[ul] |
| ha-NE | Hausa[l] | | ml | Malayalam[ul] |
| ha | Hausa | | mn | Mongolian |
| haw | Hawaiian | | mr | Marathi[ul] |
| he | Hebrew[ul] | | ms-BN | Malay[l] |
| hi | Hindi[u] | | ms-SG | Malay[l] |
| hr | Croatian[ul] | | ms | Malay[ul] |

| | | | |
|---|---|---|---|
| mt | Maltese | smn | Inari Sami |
| mua | Mundang | sn | Shona |
| my | Burmese | so | Somali |
| mzn | Mazanderani | sq | Albanian[ul] |
| naq | Nama | sr-Cyrl-BA | Serbian[ul] |
| nb | Norwegian Bokmål[ul] | sr-Cyrl-ME | Serbian[ul] |
| nd | North Ndebele | sr-Cyrl-XK | Serbian[ul] |
| ne | Nepali | sr-Cyrl | Serbian[ul] |
| nl | Dutch[ul] | sr-Latn-BA | Serbian[ul] |
| nmg | Kwasio | sr-Latn-ME | Serbian[ul] |
| nn | Norwegian Nynorsk[ul] | sr-Latn-XK | Serbian[ul] |
| nnh | Ngiemboon | sr-Latn | Serbian[ul] |
| nus | Nuer | sr | Serbian[ul] |
| nyn | Nyankole | sv | Swedish[ul] |
| om | Oromo | sw | Swahili |
| or | Odia | ta | Tamil[u] |
| os | Ossetic | te | Telugu[ul] |
| pa-Arab | Punjabi | teo | Teso |
| pa-Guru | Punjabi | th | Thai[ul] |
| pa | Punjabi | ti | Tigrinya |
| pl | Polish[ul] | tk | Turkmen[ul] |
| pms | Piedmontese[ul] | to | Tongan |
| ps | Pashto | tr | Turkish[ul] |
| pt-BR | Portuguese[ul] | twq | Tasawaq |
| pt-PT | Portuguese[ul] | tzm | Central Atlas Tamazight |
| pt | Portuguese[ul] | ug | Uyghur |
| qu | Quechua | uk | Ukrainian[ul] |
| rm | Romansh[ul] | ur | Urdu[ul] |
| rn | Rundi | uz-Arab | Uzbek |
| ro | Romanian[ul] | uz-Cyrl | Uzbek |
| rof | Rombo | uz-Latn | Uzbek |
| ru | Russian[ul] | uz | Uzbek |
| rw | Kinyarwanda | vai-Latn | Vai |
| rwk | Rwa | vai-Vaii | Vai |
| sa-Beng | Sanskrit | vai | Vai |
| sa-Deva | Sanskrit | vi | Vietnamese[ul] |
| sa-Gujr | Sanskrit | vun | Vunjo |
| sa-Knda | Sanskrit | wae | Walser |
| sa-Mlym | Sanskrit | xog | Soga |
| sa-Telu | Sanskrit | yav | Yangben |
| sa | Sanskrit | yi | Yiddish |
| sah | Sakha | yo | Yoruba |
| saq | Samburu | yue | Cantonese |
| sbp | Sangu | zgh | Standard Moroccan |
| se | Northern Sami[ul] | | Tamazight |
| seh | Sena | zh-Hans-HK | Chinese |
| ses | Koyraboro Senni | zh-Hans-MO | Chinese |
| sg | Sango | zh-Hans-SG | Chinese |
| shi-Latn | Tachelhit | zh-Hans | Chinese |
| shi-Tfng | Tachelhit | zh-Hant-HK | Chinese |
| shi | Tachelhit | zh-Hant-MO | Chinese |
| si | Sinhala | zh-Hant | Chinese |
| sk | Slovak[ul] | zh | Chinese |
| sl | Slovenian[ul] | zu | Zulu |

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

with an ldf file). These are also the names recognized by \babelprovide with a valueless import.

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

---

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

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:

LUATEX/XETEX

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

---

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

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

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

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

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

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

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

**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.

26

## 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

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`.

> **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>
<dt>intraspace=</dt>
<dd>

⟨*base*⟩ ⟨*shrink*⟩ ⟨*stretch*⟩

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

</dd>

<dt>intrapenalty=</dt>
<dd>

⟨*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).

</dd>

<dt>transforms=</dt>
<dd>

⟨*transform-list*⟩

See section 1.21.

</dd>

<dt>justification=</dt>
<dd>

kashida | elongated | unhyphenated

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

</dd>

<dt>linebreaking=</dt>
<dd>

New 3.59   Just a synonymous for justification.

</dd>

<dt>mapfont=</dt>
<dd>

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.

</dd>
</dl>

> **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}
\telugudigits{1234}
\telugucounter{section}
\end{document}
```

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`
**Bengali**  `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=..*⟩]{⟨*year*⟩}{⟨*month*⟩}{⟨*day*⟩}

By default the calendar is the Gregorian, but an ini file may define strings for other calendars (currently ar, ar-*, he, fa, hi). In the latter case, the three arguments are the year, the month, and the day in those in the corresponding calendar. They are *not* the Gregorian data to be converted (which means, say, 13 is a valid month number with calendar=hebrew).
Even with a certain calendar there may be variants. In Kurmanji the default variant prints something like *30. Çileya Pêşîn 2019*, but with variant=izafa it prints *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. 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 are 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**  To set hyphenation exceptions in the preamble before any language is explicitly set with a selector, use `\babelhyphenation` instead of `\hyphenation`. 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 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).

---

[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.

## 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 previouly 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. (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". |
| 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. |

---

[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 | `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 *æ*, *Æ*, *œ*, *Œ*. |
| Latin | `letters.noj` | Replaces *j*, *J* with *i*, *I*. |
| Latin | `letters.uv` | Replaces *v*, *U* with *u*, *V*. |
| Sanskrit | `transliteration.iast` | The IAST system to romanize Devanagari.[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 ([ïü]), the replacement could be {1|ïü|íú}, which maps *ï* to *í*, and *ü* 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:

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

There are some package options controlling bidi writing.

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

---

[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.

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-'aṣr} (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).

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   .

---

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

**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}
```

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
**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.

---

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

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

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

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

### 1.28   Unicode character properties in luatex

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

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

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

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

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

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

### 1.29   Tweaking some features

`\babeladjust`   {⟨*key-value-list*⟩}

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

### 1.30   Tips, workarounds, known issues and notes

- If you use the document class book *and* you use `\ref` inside the argument of `\chapter` (or just use `\ref` inside `\MakeUppercase`), LaTeX will keep complaining about an undefined label. To prevent such problems, you can revert to using uppercase labels, you can use `\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.
**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.

---

[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.

### 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, $\epsilon$-TeX, the main exception being luatex) require hyphenation patterns to be preloaded when a format is created (eg, LaTeX, XeLaTeX, pdfLaTeX). babel provides a tool which has become standard in many distributions and based on a "configuration file" named `language.dat`. The exact way this file is used depends on the distribution, so please, read the documentation for the latter (note also some distributions generate the file with some tool).
New 3.9q  With luatex, however, patterns are loaded on the fly when requested by the language (except the "0th" language, typically english, which is preloaded always).[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]

---

[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.
[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`.

## 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 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.

---

[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 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.

## 3.1 Guidelines for contributed languages

Currently, the easiest way to contribute a new language is by taking one the 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.

---

[27]But not removed, for backward compatibility.

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

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

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

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

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

## 3.2 Basic macros

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

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

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

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

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

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

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

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

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

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

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

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

`\main@language`  To postpone the activation of the definitions needed for a language until the beginning of a

document, all language definition files should use \main@language instead of \selectlanguage. This will just store the name of the language, and the proper language will be activated at the start of the document.

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

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

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

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

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

\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,

---

[28]This mechanism was introduced by Bernd Raichle.

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 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]
  \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}
```

---

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

```
    \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*⟩}

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*⟩}

---

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

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=3.76.2766⟩⟩
2 ⟨⟨date=2022/06/12⟩⟩

**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@csarg#1#2{\expandafter#1\csname bbl@#2\endcsname}%
12 \def\bbl@cs#1{\csname bbl@#1\endcsname}
13 \def\bbl@cl#1{\csname bbl@#1@\languagename\endcsname}
14 \def\bbl@loop#1#2#3{\bbl@@loop#1{#3}#2,\@nnil,}
15 \def\bbl@loopx#1#2{\expandafter\bbl@loop\expandafter#1\expandafter{#2}}
16 \def\bbl@@loop#1#2#3,{%
17   \ifx\@nnil#3\relax\else
18     \def#1{#3}#2\bbl@afterfi\bbl@@loop#1{#2}%
19   \fi}
20 \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.

```
21 \def\bbl@add@list#1#2{%
22   \edef#1{%
23     \bbl@ifunset{\bbl@stripslash#1}%
24       {}%
25       {\ifx#1\@empty\else#1,\fi}%
26     #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.

```
27 \long\def\bbl@afterelse#1\else#2\fi{\fi#1}
28 \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

---

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

one-level expansion (where `..` is the macro name without the backslash). The result may be followed by extra arguments, if necessary.

```
29 \def\bbl@exp#1{%
30   \begingroup
31     \let\\\noexpand
32     \let\<\bbl@exp@en
33     \let\[\bbl@exp@ue
34     \edef\bbl@exp@aux{\endgroup#1}%
35   \bbl@exp@aux}
36 \def\bbl@exp@en#1>{\expandafter\noexpand\csname#1\endcsname}%
37 \def\bbl@exp@ue#1]{%
38   \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.

```
39 \def\bbl@tempa#1{%
40   \long\def\bbl@trim##1##2{%
41     \futurelet\bbl@trim@a\bbl@trim@c##2\@nil\@nil#1\@nil\relax{##1}}%
42   \def\bbl@trim@c{%
43     \ifx\bbl@trim@a\@sptoken
44       \expandafter\bbl@trim@b
45     \else
46       \expandafter\bbl@trim@b\expandafter#1%
47     \fi}%
48   \long\def\bbl@trim@b#1##1 \@nil{\bbl@trim@i##1}}
49 \bbl@tempa{ }
50 \long\def\bbl@trim@i#1\@nil#2\relax#3{#3{#1}}
51 \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 $\epsilon$-tex engine, it is based on \ifcsname, which is more efficient, and does not waste memory.

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

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

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

For each element in the comma separated <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).

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

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

```
88 \def\bbl@vforeach#1#2{%
89   \def\bbl@forcmd##1{#2}%
90   \bbl@fornext#1,\@nil,}
91 \def\bbl@fornext#1,{%
92   \ifx\@nil#1\relax\else
93     \bbl@ifblank{#1}{}{\bbl@trim\bbl@forcmd{#1}}%
94     \expandafter\bbl@fornext
95   \fi}
96 \def\bbl@foreach#1{\expandafter\bbl@vforeach\expandafter{#1}}
```

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

```
97 \def\bbl@replace#1#2#3{% in #1 -> repl #2 by #3
98   \toks@{}%
99   \def\bbl@replace@aux##1#2##2#2{%
100    \ifx\bbl@nil##2%
101      \toks@\expandafter{\the\toks@##1}%
102    \else
103      \toks@\expandafter{\the\toks@##1#3}%
104      \bbl@afterfi
105      \bbl@replace@aux##2#2%
106    \fi}%
107    \expandafter\bbl@replace@aux#1#2\bbl@nil#2%
108    \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).

```
109 \ifx\detokenize\@undefined\else % Unused macros if old Plain TeX
110   \bbl@exp{\def\\\bbl@parsedef##1\detokenize{macro:}#2->#3\relax{%
111     \def\bbl@tempa{#1}%
112     \def\bbl@tempb{#2}%
113     \def\bbl@tempe{#3}}
114   \def\bbl@sreplace#1#2#3{%
115     \begingroup
116       \expandafter\bbl@parsedef\meaning#1\relax
117       \def\bbl@tempc{#2}%
118       \edef\bbl@tempc{\expandafter\strip@prefix\meaning\bbl@tempc}%
119       \def\bbl@tempd{#3}%
120       \edef\bbl@tempd{\expandafter\strip@prefix\meaning\bbl@tempd}%
121       \bbl@xin@{\bbl@tempc}{\bbl@tempe}% If not in macro, do nothing
122       \ifin@
123         \bbl@exp{\\\bbl@replace\\\bbl@tempe{\bbl@tempc}{\bbl@tempd}}%
124         \def\bbl@tempc{%      Expanded an executed below as 'uplevel'
```

```
125            \\\makeatletter % "internal" macros with @ are assumed
126            \\\scantokens{%
127              \bbl@tempa\\\@namedef{\bbl@stripslash#1}\bbl@tempb{\bbl@tempe}}%
128            \catcode64=\the\catcode64\relax}%  Restore @
129          \else
130            \let\bbl@tempc\@empty  % Not \relax
131          \fi
132          \bbl@exp{%       For the 'uplevel' assignments
133        \endgroup
134          \bbl@tempc}}  % empty or expand to set #1 with changes
135 \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.

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

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

```
158 \def\bbl@bsphack{%
159   \ifhmode
160     \hskip\z@skip
161     \def\bbl@esphack{\loop\ifdim\lastskip>\z@\unskip\repeat\unskip}%
162   \else
163     \let\bbl@esphack\@empty
164   \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.

```
165 \def\bbl@cased{%
166   \ifx\oe\OE
167     \expandafter\in@\expandafter
168       {\expandafter\OE\expandafter}\expandafter{\oe}%
169     \ifin@
170       \bbl@afterelse\expandafter\MakeUppercase
171     \else
172       \bbl@afterfi\expandafter\MakeLowercase
173     \fi
174   \else
175     \expandafter\@firstofone
176   \fi}
```

An alternative to \IfFormatAtLeastTF for old versions. Temporary.

```
177 \ifx\IfFormatAtLeastTF\@undefined
178   \def\bbl@ifformatlater{\@ifl@t@r\fmtversion}
179 \else
180   \let\bbl@ifformatlater\IfFormatAtLeastTF
181 \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`).

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

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

```
200 ⟨⟨*Define core switching macros⟩⟩ ≡
201 \ifx\language\@undefined
202   \csname newcount\endcsname\language
203 \fi
204 ⟨⟨/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.

```
205 ⟨⟨*Define core switching macros⟩⟩ ≡
206 \countdef\last@language=19
207 \def\addlanguage{\csname newlanguage\endcsname}
208 ⟨⟨/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`)

```
209 ⟨*package⟩
210 \NeedsTeXFormat{LaTeX2e}[2005/12/01]
211 \ProvidesPackage{babel}[⟨⟨date⟩⟩ ⟨⟨version⟩⟩ The Babel package]
```

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

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

```
248 ⟨⟨Basic macros⟩⟩
249 \@ifpackagewith{babel}{silent}
250   {\let\bbl@info\@gobble
251    \let\bbl@infowarn\@gobble
252    \let\bbl@warning\@gobble}
253   {}
254 %
255 \def\AfterBabelLanguage#1{%
256   \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.

```
257 \ifx\bbl@languages\@undefined\else
258   \begingroup
259     \catcode`\^^I=12
```

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

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

```
295 \bbl@trace{key=value and another general options}
296 \bbl@csarg\let{tempa\expandafter}\csname opt@babel.sty\endcsname
297 \def\bbl@tempb#1.#2{%  Remove trailing dot
298   #1\ifx\@empty#2\else,\bbl@afterfi\bbl@tempb#2\fi}%
299 \def\bbl@tempd#1.#2\@nnil{%  TODO. Refactor lists?
300   \ifx\@empty#2%
301     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
302   \else
303     \in@{,provide=}{,#1}%
304     \ifin@
305       \edef\bbl@tempc{%
306         \ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.\bbl@tempb#2}%
307     \else
```

```
308        \in@{=}{#1}%
309        \ifin@
310          \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.#2}%
311        \else
312          \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
313          \bbl@csarg\edef{mod@#1}{\bbl@tempb#2}%
314        \fi
315      \fi
316    \fi}
317  \let\bbl@tempc\@empty
318  \bbl@foreach\bbl@tempa{\bbl@tempd#1.\@empty\@nnil}
319  \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.

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

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

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

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

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

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

```
365 \ProcessOptions*

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

## 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=....

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

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

```
392   \def\bbl@ifshorthand#1{%
393     \bbl@xin@{\string#1}{\bbl@opt@shorthands}%
394     \ifin@
395       \expandafter\@firstoftwo
396     \else
397       \expandafter\@secondoftwo
398     \fi}
```

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

```
399   \edef\bbl@opt@shorthands{%
400     \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.

```
401   \bbl@ifshorthand{'}%
402     {\PassOptionsToPackage{activeacute}{babel}}{}
```

```
403    \bbl@ifshorthand{`}%
404      {\PassOptionsToPackage{activegrave}{babel}}{}
405 \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.

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

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

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

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

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

```
436 ⟨/core⟩
437 ⟨∗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.

```
438 \def\bbl@version{⟨⟨version⟩⟩}
439 \def\bbl@date{⟨⟨date⟩⟩}
440 ⟨⟨Define core switching macros⟩⟩
```

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.

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

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

```
474 \def\bbl@bcpcase#1#2#3#4\@@#5{%
475   \ifx\@empty#3%
476     \uppercase{\def#5{#1#2}}%
477   \else
478     \uppercase{\def#5{#1}}%
479     \lowercase{\edef#5{#5#2#3#4}}%
480   \fi}
481 \def\bbl@bcplookup#1-#2-#3-#4\@@{%
482   \let\bbl@bcp\relax
483   \lowercase{\def\bbl@tempa{#1}}%
484   \ifx\@empty#2%
485     \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
486   \else\ifx\@empty#3%
487     \bbl@bcpcase#2\@empty\@empty\@@\bbl@tempb
488     \IfFileExists{babel-\bbl@tempa-\bbl@tempb.ini}%
```

```
489        {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb}}%
490        {}%
491      \ifx\bbl@bcp\relax
492        \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
493      \fi
494    \else
495      \bbl@bcpcase#2\@empty\@empty\@@\bbl@tempb
496      \bbl@bcpcase#3\@empty\@empty\@@\bbl@tempc
497      \IfFileExists{babel-\bbl@tempa-\bbl@tempb-\bbl@tempc.ini}%
498        {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb-\bbl@tempc}}%
499        {}%
500      \ifx\bbl@bcp\relax
501        \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
502          {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
503          {}%
504      \fi
505      \ifx\bbl@bcp\relax
506        \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
507          {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
508          {}%
509      \fi
510      \ifx\bbl@bcp\relax
511        \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
512      \fi
513    \fi\fi}
514  \let\bbl@initoload\relax
515  \def\bbl@provide@locale{%
516    \ifx\babelprovide\@undefined
517      \bbl@error{For a language to be defined on the fly 'base'\\%
518                  is not enough, and the whole package must be\\%
519                  loaded. Either delete the 'base' option or\\%
520                  request the languages explicitly}%
521                {See the manual for further details.}%
522    \fi
523  % TODO. Option to search if loaded, with \LocaleForEach
524    \let\bbl@auxname\languagename % Still necessary. TODO
525    \bbl@ifunset{bbl@bcp@map@\languagename}{}% Move uplevel??
526      {\edef\languagename{\@nameuse{bbl@bcp@map@\languagename}}}%
527    \ifbbl@bcpallowed
528      \expandafter\ifx\csname date\languagename\endcsname\relax
529        \expandafter
530        \bbl@bcplookup\languagename-\@empty-\@empty-\@empty\@@
531        \ifx\bbl@bcp\relax\else  % Returned by \bbl@bcplookup
532          \edef\languagename{\bbl@bcp@prefix\bbl@bcp}%
533          \edef\localename{\bbl@bcp@prefix\bbl@bcp}%
534          \expandafter\ifx\csname date\languagename\endcsname\relax
535            \let\bbl@initoload\bbl@bcp
536            \bbl@exp{\\\babelprovide[\bbl@autoload@bcpoptions]{\languagename}}%
537            \let\bbl@initoload\relax
538          \fi
539          \bbl@csarg\xdef{bcp@map@\bbl@bcp}{\localename}%
540        \fi
541      \fi
542    \fi
543    \expandafter\ifx\csname date\languagename\endcsname\relax
544      \IfFileExists{babel-\languagename.tex}%
545        {\bbl@exp{\\\babelprovide[\bbl@autoload@options]{\languagename}}}%
546        {}%
547    \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.

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

```
555 \let\bbl@select@type\z@
556 \edef\selectlanguage{%
557   \noexpand\protect
558   \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.

```
559 \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.

```
560 \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.

```
561 \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
```
562 \def\bbl@push@language{%
563   \ifx\languagename\@undefined\else
564     \ifx\currentgrouplevel\@undefined
565       \xdef\bbl@language@stack{\languagename+\bbl@language@stack}%
566     \else
567       \ifnum\currentgrouplevel=\z@
568         \xdef\bbl@language@stack{\languagename+}%
569       \else
570         \xdef\bbl@language@stack{\languagename+\bbl@language@stack}%
571       \fi
572     \fi
573   \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.

```
574 \def\bbl@pop@lang#1+#2\@@{%
575   \edef\languagename{#1}%
576   \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).

```
577 \let\bbl@ifrestoring\@secondoftwo
578 \def\bbl@pop@language{%
579   \expandafter\bbl@pop@lang\bbl@language@stack\@@
580   \let\bbl@ifrestoring\@firstoftwo
581   \expandafter\bbl@set@language\expandafter{\languagename}%
582   \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).

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

The unprotected part of \selectlanguage.

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

**\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).

74

```
606 \def\BabelContentsFiles{toc,lof,lot}
607 \def\bbl@set@language#1{% from selectlanguage, pop@
608   % The old buggy way. Preserved for compatibility.
609   \edef\languagename{%
610     \ifnum\escapechar=\expandafter`\string#1\@empty
611     \else\string#1\@empty\fi}%
612   \ifcat\relax\noexpand#1%
613     \expandafter\ifx\csname date\languagename\endcsname\relax
614       \edef\languagename{#1}%
615       \let\localename\languagename
616     \else
617       \bbl@info{Using '\string\language' instead of 'language' is\\%
618                 deprecated. If what you want is to use a\\%
619                 macro containing the actual locale, make\\%
620                 sure it does not not match any language.\\%
621                 Reported}%
622       \ifx\scantokens\@undefined
623         \def\localename{??}%
624       \else
625         \scantokens\expandafter{\expandafter
626           \def\expandafter\localename\expandafter{\languagename}}%
627       \fi
628     \fi
629   \else
630     \def\localename{#1}% This one has the correct catcodes
631   \fi
632   \select@language{\languagename}%
633   % write to auxs
634   \expandafter\ifx\csname date\languagename\endcsname\relax\else
635     \if@filesw
636       \ifx\babel@aux\@gobbletwo\else % Set if single in the first, redundant
637         \bbl@savelastskip
638         \protected@write\@auxout{}{\string\babel@aux{\bbl@auxname}{}}%
639         \bbl@restorelastskip
640       \fi
641       \bbl@usehooks{write}{}%
642     \fi
643   \fi}
644 %
645 \let\bbl@restorelastskip\relax
646 \let\bbl@savelastskip\relax
647 %
648 \newif\ifbbl@bcpallowed
649 \bbl@bcpallowedfalse
650 \def\select@language#1{% from set@, babel@aux
651   \ifx\bbl@selectorname\@empty
652     \def\bbl@selectorname{select}%
653   % set hymap
654   \fi
655   \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
656   % set name
657   \edef\languagename{#1}%
658   \bbl@fixname\languagename
659   % TODO. name@map must be here?
660   \bbl@provide@locale
661   \bbl@iflanguage\languagename{%
662     \expandafter\ifx\csname date\languagename\endcsname\relax
663       \bbl@error
664         {Unknown language '\languagename'. Either you have\\%
665         misspelled its name, it has not been installed,\\%
666         or you requested it in a previous run. Fix its name,\\%
667         install it or just rerun the file, respectively. In\\%
668         some cases, you may need to remove the aux file}%
```

```
669          {You may proceed, but expect wrong results}%
670       \else
671         % set type
672         \let\bbl@select@type\z@
673         \expandafter\bbl@switch\expandafter{\languagename}%
674      \fi}}
675 \def\babel@aux#1#2{%
676   \select@language{#1}%
677   \bbl@foreach\BabelContentsFiles{% \relax -> don't assume vertical mode
678      \@writefile{##1}{\babel@toc{#1}{#2}\relax}}}% TODO - plain?
679 \def\babel@toc#1#2{%
680   \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.

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

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

otherlanguage    The otherlanguage environment can be used as an alternative to using the \selectlanguage
declarative command. When you are typesetting a document which mixes left-to-right and
right-to-left typesetting you have to use this environment in order to let things work as you expect
them to.
The \ignorespaces command is necessary to hide the environment when it is entered in horizontal
mode.

```
766 \long\def\otherlanguage#1{%
767   \def\bbl@selectorname{other}%
768   \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\thr@@\fi
769   \csname selectlanguage \endcsname{#1}%
770   \ignorespaces}
```

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

```
771 \long\def\endotherlanguage{%
```

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

```
773 \expandafter\def\csname otherlanguage*\endcsname{%
774   \@ifnextchar[\bbl@otherlanguage@s{\bbl@otherlanguage@s[]}}
775 \def\bbl@otherlanguage@s[#1]#2{%
776   \def\bbl@selectorname{other*}%
777   \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
778   \def\bbl@select@opts{#1}%
779   \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".

```
780 \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.

```
781 \providecommand\bbl@beforeforeign{}
782 \edef\foreignlanguage{%
783   \noexpand\protect
784   \expandafter\noexpand\csname foreignlanguage \endcsname}
785 \expandafter\def\csname foreignlanguage \endcsname{%
786   \@ifstar\bbl@foreign@s\bbl@foreign@x}
787 \providecommand\bbl@foreign@x[3][]{%
788   \begingroup
789     \def\bbl@selectorname{foreign}%
790     \def\bbl@select@opts{#1}%
791     \let\BabelText\@firstofone
792     \bbl@beforeforeign
793     \foreign@language{#2}%
794     \bbl@usehooks{foreign}{}%
795     \BabelText{#3}% Now in horizontal mode!
796   \endgroup}
797 \def\bbl@foreign@s#1#2{% TODO - \shapemode, \@setpar, ?\@@par
798   \begingroup
799     {\par}%
800     \def\bbl@selectorname{foreign*}%
801     \let\bbl@select@opts\@empty
802     \let\BabelText\@firstofone
803     \foreign@language{#1}%
804     \bbl@usehooks{foreign*}{}%
805     \bbl@dirparastext
```

```
806     \BabelText{#2}% Still in vertical mode!
807   {\par}%
808   \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.

```
809 \def\foreign@language#1{%
810   % set name
811   \edef\languagename{#1}%
812   \ifbbl@usedategroup
813     \bbl@add\bbl@select@opts{,date,}%
814     \bbl@usedategroupfalse
815   \fi
816   \bbl@fixname\languagename
817   % TODO. name@map here?
818   \bbl@provide@locale
819   \bbl@iflanguage\languagename{%
820     \expandafter\ifx\csname date\languagename\endcsname\relax
821       \bbl@warning   % TODO - why a warning, not an error?
822         {Unknown language '#1'. Either you have\\%
823          misspelled its name, it has not been installed,\\%
824          or you requested it in a previous run. Fix its name,\\%
825          install it or just rerun the file, respectively. In\\%
826          some cases, you may need to remove the aux file.\\%
827          I'll proceed, but expect wrong results.\\%
828          Reported}%
829     \fi
830     % set type
831     \let\bbl@select@type\@ne
832     \expandafter\bbl@switch\expandafter{\languagename}}}
```

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

```
833 \def\IfBabelSelectorTF#1{%
834   \bbl@xin@{,\bbl@selectorname,}{,\zap@space#1 \@empty,}%
835   \ifin@
836     \expandafter\@firstoftwo
837   \else
838     \expandafter\@secondoftwo
839   \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.

```
840 \let\bbl@hyphlist\@empty
841 \let\bbl@hyphenation@\relax
842 \let\bbl@pttnlist\@empty
843 \let\bbl@patterns@\relax
844 \let\bbl@hymapsel=\@cclv
845 \def\bbl@patterns#1{%
846   \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
847       \csname l@#1\endcsname
848       \edef\bbl@tempa{#1}%
849     \else
850       \csname l@#1:\f@encoding\endcsname
851       \edef\bbl@tempa{#1:\f@encoding}%
852     \fi
853   \@expandtwoargs\bbl@usehooks{patterns}{{#1}{\bbl@tempa}}%
```

```
854   %  > luatex
855   \@ifundefined{bbl@hyphenation@}{}{% Can be \relax!
856     \begingroup
857       \bbl@xin@{,\number\language,}{,\bbl@hyphlist}%
858       \ifin@\else
859         \@expandtwoargs\bbl@usehooks{hyphenation}{{#1}{\bbl@tempa}}%
860         \hyphenation{%
861           \bbl@hyphenation@
862           \@ifundefined{bbl@hyphenation@#1}%
863             \@empty
864             {\space\csname bbl@hyphenation@#1\endcsname}}%
865         \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
866       \fi
867     \endgroup}}
```

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

```
868 \def\hyphenrules#1{%
869   \edef\bbl@tempf{#1}%
870   \bbl@fixname\bbl@tempf
871   \bbl@iflanguage\bbl@tempf{%
872     \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
873     \ifx\languageshorthands\@undefined\else
874       \languageshorthands{none}%
875     \fi
876     \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax
877       \set@hyphenmins\tw@\thr@@\relax
878     \else
879       \expandafter\expandafter\expandafter\set@hyphenmins
880       \csname\bbl@tempf hyphenmins\endcsname\relax
881     \fi}}
882 \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.

```
883 \def\providehyphenmins#1#2{%
884   \expandafter\ifx\csname #1hyphenmins\endcsname\relax
885     \@namedef{#1hyphenmins}{#2}%
886   \fi}
```

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

```
887 \def\set@hyphenmins#1#2{%
888   \lefthyphenmin#1\relax
889   \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.

```
890 \ifx\ProvidesFile\@undefined
891   \def\ProvidesLanguage#1[#2 #3 #4]{%
892     \wlog{Language: #1 #4 #3 <#2>}%
893     }
894 \else
895   \def\ProvidesLanguage#1{%
896     \begingroup
897       \catcode`\ 10 %
898       \@makeother\/%
```

```
899      \@ifnextchar[%]
900        {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}}
901    \def\@provideslanguage#1[#2]{%
902      \wlog{Language: #1 #2}%
903      \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
904      \endgroup}
905 \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.

```
906 \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.

```
907 \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':

```
908 \providecommand\setlocale{%
909    \bbl@error
910      {Not yet available}%
911      {Find an armchair, sit down and wait}}
912 \let\uselocale\setlocale
913 \let\locale\setlocale
914 \let\selectlocale\setlocale
915 \let\textlocale\setlocale
916 \let\textlanguage\setlocale
917 \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$_\varepsilon$, 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.

```
918 \edef\bbl@nulllanguage{\string\language=0}
919 \def\bbl@nocaption{\protect\bbl@nocaption@i}
920 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
921    \global\@namedef{#2}{\textbf{?#1?}}%
922    \@nameuse{#2}%
923    \edef\bbl@tempa{#1}%
924    \bbl@sreplace\bbl@tempa{name}{}%
925    \bbl@warning{% TODO.
926      \@backslashchar#1 not set for '\languagename'. Please,\\%
927      define it after the language has been loaded\\%
928      (typically in the preamble) with:\\%
929      \string\setlocalecaption{\languagename}{\bbl@tempa}{..}\\%
930      Reported}}
931 \def\bbl@tentative{\protect\bbl@tentative@i}
932 \def\bbl@tentative@i#1{%
933    \bbl@warning{%
934      Some functions for '#1' are tentative.\\%
935      They might not work as expected and their behavior\\%
936      could change in the future.\\%
937      Reported}}
938 \def\@nolanerr#1{%
939    \bbl@error
```

```
940     {You haven't defined the language '#1' yet.\\%
941      Perhaps you misspelled it or your installation\\%
942      is not complete}%
943     {Your command will be ignored, type <return> to proceed}}
944 \def\@nopatterns#1{%
945   \bbl@warning
946     {No hyphenation patterns were preloaded for\\%
947      the language '#1' into the format.\\%
948      Please, configure your TeX system to add them and\\%
949      rebuild the format. Now I will use the patterns\\%
950      preloaded for \bbl@nulllanguage\space instead}}
951 \let\bbl@usehooks\@gobbletwo
952 \ifx\bbl@onlyswitch\@empty\endinput\fi
953  % Here ended switch.def
```

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

```
954 \ifx\directlua\@undefined\else
955   \ifx\bbl@luapatterns\@undefined
956     \input luababel.def
957   \fi
958 \fi
959 ⟨⟨Basic macros⟩⟩
960 \bbl@trace{Compatibility with language.def}
961 \ifx\bbl@languages\@undefined
962   \ifx\directlua\@undefined
963     \openin1 = language.def % TODO. Remove hardcoded number
964     \ifeof1
965       \closein1
966       \message{I couldn't find the file language.def}
967     \else
968       \closein1
969       \begingroup
970         \def\addlanguage#1#2#3#4#5{%
971           \expandafter\ifx\csname lang@#1\endcsname\relax\else
972             \global\expandafter\let\csname l@#1\expandafter\endcsname
973               \csname lang@#1\endcsname
974           \fi}%
975         \def\uselanguage#1{}%
976         \input language.def
977       \endgroup
978     \fi
979   \fi
980   \chardef\l@english\z@
981 \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.

```
982 \def\addto#1#2{%
983   \ifx#1\@undefined
984     \def#1{#2}%
985   \else
986     \ifx#1\relax
987       \def#1{#2}%
988     \else
989       {\toks@\expandafter{#1#2}%
990        \xdef#1{\the\toks@}}%
991     \fi
992   \fi}
```

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

```
993 \def\bbl@withactive#1#2{%
994   \begingroup
995     \lccode`\~=`#2\relax
996     \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.

```
997 \def\bbl@redefine#1{%
998   \edef\bbl@tempa{\bbl@stripslash#1}%
999   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1000   \expandafter\def\csname\bbl@tempa\endcsname}
1001 \@onlypreamble\bbl@redefine
```

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

```
1002 \def\bbl@redefine@long#1{%
1003   \edef\bbl@tempa{\bbl@stripslash#1}%
1004   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1005   \expandafter\long\expandafter\def\csname\bbl@tempa\endcsname}
1006 \@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␣.

```
1007 \def\bbl@redefinerobust#1{%
1008   \edef\bbl@tempa{\bbl@stripslash#1}%
1009   \bbl@ifunset{\bbl@tempa\space}%
1010     {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
1011      \bbl@exp{\def\\#1{\\\protect\<\bbl@tempa\space>}}}%
1012     {\bbl@exp{\let\<org@\bbl@tempa>\<\bbl@tempa\space>}}%
1013   \@namedef{\bbl@tempa\space}}
1014 \@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.

```
1015 \bbl@trace{Hooks}
1016 \newcommand\AddBabelHook[3][]{%
1017   \bbl@ifunset{bbl@hk@#2}{\EnableBabelHook{#2}}{}%
1018   \def\bbl@tempa##1,#3=##2,##3\@empty{\def\bbl@tempb{##2}}%
1019   \expandafter\bbl@tempa\bbl@evargs,#3=,\@empty
1020   \bbl@ifunset{bbl@ev@#2@#3@#1}%
1021     {\bbl@csarg\bbl@add{ev@#3@#1}{\bbl@elth{#2}}}%
1022     {\bbl@csarg\let{ev@#2@#3@#1}\relax}%
1023   \bbl@csarg\newcommand{ev@#2@#3@#1}[\bbl@tempb]}
1024 \newcommand\EnableBabelHook[1]{\bbl@csarg\let{hk@#1}\@firstofone}
1025 \newcommand\DisableBabelHook[1]{\bbl@csarg\let{hk@#1}\@gobble}
1026 \def\bbl@usehooks#1#2{%
1027   \ifx\UseHook\@undefined\else\UseHook{babel/*/#1}\fi
1028   \def\bbl@elth##1{%
1029     \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#1@}#2}}%
1030   \bbl@cs{ev@#1@}%
1031   \ifx\languagename\@undefined\else % Test required for Plain (?)
1032     \ifx\UseHook\@undefined\else\UseHook{babel/\languagename/#1}\fi
1033     \def\bbl@elth##1{%
1034       \bbl@cs{hk@##1}{\bbl@cl{ev@##1@#1}#2}}%
1035     \bbl@cl{ev@#1}%
1036   \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).

```
1037 \def\bbl@evargs{,% <- don't delete this comma
1038   everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
1039   adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
1040   beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
1041   hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%
1042   beforestart=0,languagename=2}
1043 \ifx\NewHook\@undefined\else
1044   \def\bbl@tempa#1=#2\@@{\NewHook{babel/#1}}
1045   \bbl@foreach\bbl@evargs{\bbl@tempa#1\@@}
1046 \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.

```
1047 \bbl@trace{Defining babelensure}
1048 \newcommand\babelensure[2][]{%  TODO - revise test files
1049   \AddBabelHook{babel-ensure}{afterextras}{%
1050     \ifcase\bbl@select@type
1051       \bbl@cl{e}%
1052     \fi}%
1053   \begingroup
1054     \let\bbl@ens@include\@empty
1055     \let\bbl@ens@exclude\@empty
1056     \def\bbl@ens@fontenc{\relax}%
1057     \def\bbl@tempb##1{%
1058       \ifx\@empty##1\else\noexpand##1\expandafter\bbl@tempb\fi}%
1059     \edef\bbl@tempa{\bbl@tempb#1\@empty}%
1060     \def\bbl@tempb##1=##2\@@{\@namedef{bbl@ens@##1}{##2}}%
1061     \bbl@foreach\bbl@tempa{\bbl@tempb##1\@@}%
1062     \def\bbl@tempc{\bbl@ensure}%
1063     \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1064       \expandafter{\bbl@ens@include}}%
1065     \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1066       \expandafter{\bbl@ens@exclude}}%
1067     \toks@\expandafter{\bbl@tempc}%
1068     \bbl@exp{%
1069   \endgroup
1070   \def\<bbl@e@#2>{\the\toks@{\bbl@ens@fontenc}}}}
1071 \def\bbl@ensure#1#2#3{% 1: include 2: exclude 3: fontenc
1072   \def\bbl@tempb##1{% elt for (excluding) \bbl@captionslist list
1073     \ifx##1\@undefined % 3.32 - Don't assume the macro exists
1074       \edef##1{\noexpand\bbl@nocaption
1075         {\bbl@stripslash##1}{\languagename\bbl@stripslash##1}}%
1076     \fi
1077     \ifx##1\@empty\else
1078       \in@{##1}{#2}%
1079       \ifin@\else
1080         \bbl@ifunset{bbl@ensure@\languagename}%
1081           {\bbl@exp{%
1082             \\\DeclareRobustCommand\<bbl@ensure@\languagename>[1]{%
1083               \\\foreignlanguage{\languagename}%
1084               {\ifx\relax#3\else
1085                 \\\fontencoding{#3}\\\selectfont
1086               \fi
```

```
1087                ########1}}}}%
1088          {}%
1089        \toks@\expandafter{##1}%
1090        \edef##1{%
1091            \bbl@csarg\noexpand{ensure@\languagename}%
1092            {\the\toks@}}%
1093      \fi
1094      \expandafter\bbl@tempb
1095    \fi}%
1096  \expandafter\bbl@tempb\bbl@captionslist\today\@empty
1097  \def\bbl@tempa##1{% elt for include list
1098    \ifx##1\@empty\else
1099      \bbl@csarg\in@{ensure@\languagename\expandafter}\expandafter{##1}%
1100      \ifin@\else
1101        \bbl@tempb##1\@empty
1102      \fi
1103      \expandafter\bbl@tempa
1104    \fi}%
1105  \bbl@tempa#1\@empty}
1106 \def\bbl@captionslist{%
1107   \prefacename\refname\abstractname\bibname\chaptername\appendixname
1108   \contentsname\listfigurename\listtablename\indexname\figurename
1109   \tablename\partname\enclname\ccname\headtoname\pagename\seename
1110   \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.

```
1111 \bbl@trace{Macros for setting language files up}
1112 \def\bbl@ldfinit{%
1113   \let\bbl@screset\@empty
1114   \let\BabelStrings\bbl@opt@string
1115   \let\BabelOptions\@empty
1116   \let\BabelLanguages\relax
1117   \ifx\originalTeX\@undefined
1118     \let\originalTeX\@empty
1119   \else
1120     \originalTeX
1121   \fi}
1122 \def\LdfInit#1#2{%
1123   \chardef\atcatcode=\catcode`\@
1124   \catcode`\@=11\relax
1125   \chardef\eqcatcode=\catcode`\=
1126   \catcode`\==12\relax
1127   \expandafter\if\expandafter\@backslashchar
1128                  \expandafter\@car\string#2\@nil
```

```
1129        \ifx#2\@undefined\else
1130          \ldf@quit{#1}%
1131        \fi
1132      \else
1133        \expandafter\ifx\csname#2\endcsname\relax\else
1134          \ldf@quit{#1}%
1135        \fi
1136      \fi
1137      \bbl@ldfinit}
```

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

```
1138 \def\ldf@quit#1{%
1139    \expandafter\main@language\expandafter{#1}%
1140    \catcode`\@=\atcatcode \let\atcatcode\relax
1141    \catcode`\==\eqcatcode \let\eqcatcode\relax
1142    \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.

```
1143 \def\bbl@afterldf#1{% TODO. Merge into the next macro? Unused elsewhere
1144    \bbl@afterlang
1145    \let\bbl@afterlang\relax
1146    \let\BabelModifiers\relax
1147    \let\bbl@screset\relax}%
1148 \def\ldf@finish#1{%
1149    \loadlocalcfg{#1}%
1150    \bbl@afterldf{#1}%
1151    \expandafter\main@language\expandafter{#1}%
1152    \catcode`\@=\atcatcode \let\atcatcode\relax
1153    \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.

```
1154 \@onlypreamble\LdfInit
1155 \@onlypreamble\ldf@quit
1156 \@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.

```
1157 \def\main@language#1{%
1158    \def\bbl@main@language{#1}%
1159    \let\languagename\bbl@main@language % TODO. Set localename
1160    \bbl@id@assign
1161    \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.

```
1162 \def\bbl@beforestart{%
1163    \def\@nolanerr##1{%
1164      \bbl@warning{Undefined language '##1' in aux.\\Reported}}%
1165    \bbl@usehooks{beforestart}{}%
1166    \global\let\bbl@beforestart\relax}
1167 \AtBeginDocument{%
1168    {\@nameuse{bbl@beforestart}}%  Group!
1169    \if@filesw
1170      \providecommand\babel@aux[2]{}%
1171      \immediate\write\@mainaux{%
1172        \string\providecommand\string\babel@aux[2]{}}%
```

```
1173       \immediate\write\@mainaux{\string\@nameuse{bbl@beforestart}}%
1174    \fi
1175    \expandafter\selectlanguage\expandafter{\bbl@main@language}%
1176    \ifbbl@single  % must go after the line above.
1177      \renewcommand\selectlanguage[1]{}%
1178      \renewcommand\foreignlanguage[2]{#2}%
1179      \global\let\babel@aux\@gobbletwo  % Also as flag
1180    \fi
1181    \ifcase\bbl@engine\or\pagedir\bodydir\fi} % TODO - a better place
```

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

```
1182 \def\select@language@x#1{%
1183    \ifcase\bbl@select@type
1184      \bbl@ifsamestring\languagename{#1}{}{\select@language{#1}}%
1185    \else
1186      \select@language{#1}%
1187    \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.

```
1188 \bbl@trace{Shorhands}
1189 \def\bbl@add@special#1{% 1:a macro like \", \?, etc.
1190    \bbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
1191    \bbl@ifunset{@sanitize}{}{\bbl@add\@sanitize{\@makeother#1}}%
1192    \ifx\nfss@catcodes\@undefined\else % TODO - same for above
1193      \begingroup
1194        \catcode`#1\active
1195        \nfss@catcodes
1196        \ifnum\catcode`#1=\active
1197          \endgroup
1198          \bbl@add\nfss@catcodes{\@makeother#1}%
1199        \else
1200          \endgroup
1201        \fi
1202    \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.

```
1203 \def\bbl@remove@special#1{%
1204    \begingroup
1205      \def\x##1##2{\ifnum`#1=`##2\noexpand\@empty
1206                  \else\noexpand##1\noexpand##2\fi}%
1207      \def\do{\x\do}%
1208      \def\@makeother{\x\@makeother}%
1209    \edef\x{\endgroup
1210      \def\noexpand\dospecials{\dospecials}%
1211      \expandafter\ifx\csname @sanitize\endcsname\relax\else
1212        \def\noexpand\@sanitize{\@sanitize}%
1213      \fi}%
1214    \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`).

```
1215 \def\bbl@active@def#1#2#3#4{%
1216   \@namedef{#3#1}{%
1217     \expandafter\ifx\csname#2@sh@#1@\endcsname\relax
1218       \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
1219     \else
1220       \bbl@afterfi\csname#2@sh@#1@\endcsname
1221     \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.

```
1222   \long\@namedef{#3@arg#1}##1{%
1223     \expandafter\ifx\csname#2@sh@#1@\string##1@\endcsname\relax
1224       \bbl@afterelse\csname#4#1\endcsname##1%
1225     \else
1226       \bbl@afterfi\csname#2@sh@#1@\string##1@\endcsname
1227     \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.

```
1228 \def\initiate@active@char#1{%
1229   \bbl@ifunset{active@char\string#1}%
1230     {\bbl@withactive
1231       {\expandafter\@initiate@active@char\expandafter}#1\string#1#1}%
1232     {}}
```

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).

```
1233 \def\@initiate@active@char#1#2#3{%
1234   \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
1235   \ifx#1\@undefined
1236     \bbl@csarg\def{oridef@#2}{\def#1{\active@prefix#1\@undefined}}%
1237   \else
1238     \bbl@csarg\let{oridef@@#2}#1%
1239     \bbl@csarg\edef{oridef@#2}{%
1240       \let\noexpand#1%
1241       \expandafter\noexpand\csname bbl@oridef@@#2\endcsname}%
1242   \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*).

```
1243   \ifx#1#3\relax
1244     \expandafter\let\csname normal@char#2\endcsname#3%
1245   \else
1246     \bbl@info{Making #2 an active character}%
1247     \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
1248       \@namedef{normal@char#2}{%
```

```
1249        \textormath{#3}{\csname bbl@oridef@@#2\endcsname}}%
1250    \else
1251      \@namedef{normal@char#2}{#3}%
1252    \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).

```
1253    \bbl@restoreactive{#2}%
1254    \AtBeginDocument{%
1255      \catcode`#2\active
1256      \if@filesw
1257        \immediate\write\@mainaux{\catcode`\string#2\active}%
1258      \fi}%
1259    \expandafter\bbl@add@special\csname#2\endcsname
1260    \catcode`#2\active
1261  \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*⟩).

```
1262  \let\bbl@tempa\@firstoftwo
1263  \if\string^#2%
1264    \def\bbl@tempa{\noexpand\textormath}%
1265  \else
1266    \ifx\bbl@mathnormal\@undefined\else
1267      \let\bbl@tempa\bbl@mathnormal
1268    \fi
1269  \fi
1270  \expandafter\edef\csname active@char#2\endcsname{%
1271    \bbl@tempa
1272      {\noexpand\if@safe@actives
1273        \noexpand\expandafter
1274        \expandafter\noexpand\csname normal@char#2\endcsname
1275      \noexpand\else
1276        \noexpand\expandafter
1277        \expandafter\noexpand\csname bbl@doactive#2\endcsname
1278      \noexpand\fi}%
1279      {\expandafter\noexpand\csname normal@char#2\endcsname}}%
1280  \bbl@csarg\edef{doactive#2}{%
1281    \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!).

```
1282    \bbl@csarg\edef{active@#2}{%
1283      \noexpand\active@prefix\noexpand#1%
1284      \expandafter\noexpand\csname active@char#2\endcsname}%
1285    \bbl@csarg\edef{normal@#2}{%
1286      \noexpand\active@prefix\noexpand#1%
1287      \expandafter\noexpand\csname normal@char#2\endcsname}%
1288  \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.

```
1289    \bbl@active@def#2\user@group{user@active}{language@active}%
1290    \bbl@active@def#2\language@group{language@active}{system@active}%
1291    \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.

```
1292    \expandafter\edef\csname\user@group @sh@#2@@\endcsname
1293      {\expandafter\noexpand\csname normal@char#2\endcsname}%
1294    \expandafter\edef\csname\user@group @sh@#2@\string\protect@\endcsname
1295      {\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.

```
1296    \if\string'#2%
1297      \let\prim@s\bbl@prim@s
1298      \let\active@math@prime#1%
1299    \fi
1300    \bbl@usehooks{initiateactive}{{#1}{#2}{#3}}}
```

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

```
1301 ⟨⟨*More package options⟩⟩ ≡
1302 \DeclareOption{math=active}{}
1303 \DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
1304 ⟨⟨/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`.

```
1305 \@ifpackagewith{babel}{KeepShorthandsActive}%
1306   {\let\bbl@restoreactive\@gobble}%
1307   {\def\bbl@restoreactive#1{%
1308      \bbl@exp{%
1309        \\\AfterBabelLanguage\\\CurrentOption
1310          {\catcode`#1=\the\catcode`#1\relax}%
1311        \\\AtEndOfPackage
1312          {\catcode`#1=\the\catcode`#1\relax}}}%
1313    \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.

```
1314 \def\bbl@sh@select#1#2{%
1315   \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
1316     \bbl@afterelse\bbl@scndcs
1317   \else
1318     \bbl@afterfi\csname#1@sh@#2@sel\endcsname
1319   \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.

```
1320 \begingroup
```

```
1321 \bbl@ifunset{ifincsname}% TODO. Ugly. Correct? Only Plain?
1322   {\gdef\active@prefix#1{%
1323     \ifx\protect\@typeset@protect
1324     \else
1325       \ifx\protect\@unexpandable@protect
1326         \noexpand#1%
1327       \else
1328         \protect#1%
1329       \fi
1330       \expandafter\@gobble
1331     \fi}}
1332   {\gdef\active@prefix#1{%
1333     \ifincsname
1334       \string#1%
1335       \expandafter\@gobble
1336     \else
1337       \ifx\protect\@typeset@protect
1338       \else
1339         \ifx\protect\@unexpandable@protect
1340           \noexpand#1%
1341         \else
1342           \protect#1%
1343         \fi
1344         \expandafter\expandafter\expandafter\@gobble
1345       \fi
1346     \fi}}
1347 \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⟩.

```
1348 \newif\if@safe@actives
1349 \@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.

```
1350 \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.

```
1351 \chardef\bbl@activated\z@
1352 \def\bbl@activate#1{%
1353   \chardef\bbl@activated\@ne
1354   \bbl@withactive{\expandafter\let\expandafter}#1%
1355     \csname bbl@active@\string#1\endcsname}
1356 \def\bbl@deactivate#1{%
1357   \chardef\bbl@activated\tw@
1358   \bbl@withactive{\expandafter\let\expandafter}#1%
1359     \csname bbl@normal@\string#1\endcsname}
```

\bbl@firstcs    These macros are used only as a trick when declaring shorthands.
\bbl@scndcs
```
1360 \def\bbl@firstcs#1#2{\csname#1\endcsname}
1361 \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.

91

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.

```
1362 \def\babel@texpdf#1#2#3#4{%
1363   \ifx\texorpdfstring\@undefined
1364     \textormath{#1}{#3}%
1365   \else
1366     \texorpdfstring{\textormath{#1}{#3}}{#2}%
1367     % \texorpdfstring{\textormath{#1}{#3}}{\textormath{#2}{#4}}%
1368   \fi}
1369 %
1370 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
1371 \def\@decl@short#1#2#3\@nil#4{%
1372   \def\bbl@tempa{#3}%
1373   \ifx\bbl@tempa\@empty
1374     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
1375     \bbl@ifunset{#1@sh@\string#2@}{}%
1376       {\def\bbl@tempa{#4}%
1377        \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbl@tempa
1378        \else
1379          \bbl@info
1380            {Redefining #1 shorthand \string#2\\%
1381             in language \CurrentOption}%
1382        \fi}%
1383     \@namedef{#1@sh@\string#2@}{#4}%
1384   \else
1385     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs
1386     \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%
1387       {\def\bbl@tempa{#4}%
1388        \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
1389        \else
1390          \bbl@info
1391            {Redefining #1 shorthand \string#2\string#3\\%
1392             in language \CurrentOption}%
1393        \fi}%
1394     \@namedef{#1@sh@\string#2@\string#3@}{#4}%
1395   \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.

```
1396 \def\textormath{%
1397   \ifmmode
1398     \expandafter\@secondoftwo
1399   \else
1400     \expandafter\@firstoftwo
1401   \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'.

```
1402 \def\user@group{user}
1403 \def\language@group{english} % TODO. I don't like defaults
1404 \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.

```
1405 \def\useshorthands{%
1406   \@ifstar\bbl@usesh@s{\bbl@usesh@x{}}}
1407 \def\bbl@usesh@s#1{%
```

```
1408    \bbl@usesh@x
1409      {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bbl@activate{#1}}}%
1410      {#1}}
1411 \def\bbl@usesh@x#1#2{%
1412    \bbl@ifshorthand{#2}%
1413      {\def\user@group{user}%
1414       \initiate@active@char{#2}%
1415       #1%
1416       \bbl@activate{#2}}%
1417      {\bbl@error
1418        {I can't declare a shorthand turned off (\string#2)}
1419        {Sorry, but you can't use shorthands which have been\\%
1420         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.

```
1421 \def\user@language@group{user@\language@group}
1422 \def\bbl@set@user@generic#1#2{%
1423    \bbl@ifunset{user@generic@active#1}%
1424      {\bbl@active@def#1\user@language@group{user@active}{user@generic@active}%
1425       \bbl@active@def#1\user@group{user@generic@active}{language@active}%
1426       \expandafter\edef\csname#2@sh@#1@@\endcsname{%
1427         \expandafter\noexpand\csname normal@char#1\endcsname}%
1428       \expandafter\edef\csname#2@sh@#1@\string\protect@\endcsname{%
1429         \expandafter\noexpand\csname user@active#1\endcsname}}%
1430    \@empty}
1431 \newcommand\defineshorthand[3][user]{%
1432    \edef\bbl@tempa{\zap@space#1 \@empty}%
1433    \bbl@for\bbl@tempb\bbl@tempa{%
1434      \if*\expandafter\@car\bbl@tempb\@nil
1435        \edef\bbl@tempb{user@\expandafter\@gobble\bbl@tempb}%
1436        \@expandtwoargs
1437          \bbl@set@user@generic{\expandafter\string\@car#2\@nil}\bbl@tempb
1438      \fi
1439      \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].

```
1440 \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".

```
1441 \def\aliasshorthand#1#2{%
1442    \bbl@ifshorthand{#2}%
1443      {\expandafter\ifx\csname active@char\string#2\endcsname\relax
1444         \ifx\document\@notprerr
1445           \@notshorthand{#2}%
1446         \else
1447           \initiate@active@char{#2}%
1448           \expandafter\let\csname active@char\string#2\expandafter\endcsname
1449             \csname active@char\string#1\endcsname
1450           \expandafter\let\csname normal@char\string#2\expandafter\endcsname
1451             \csname normal@char\string#1\endcsname
1452           \bbl@activate{#2}%
1453         \fi
1454      \fi}%
1455      {\bbl@error
1456        {Cannot declare a shorthand turned off (\string#2)}
```

93

```
1457            {Sorry, but you cannot use shorthands which have been\\%
1458             turned off in the package options}}}
```

```
1459 \def\@notshorthand#1{%
1460   \bbl@error{%
1461     The character '\string #1' should be made a shorthand character;\\%
1462     add the command \string\useshorthands\string{#1\string} to
1463     the preamble.\\%
1464     I will ignore your instruction}%
1465   {You may proceed, but expect unexpected results}}
```

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

```
1466 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
1467 \DeclareRobustCommand*\shorthandoff{%
1468   \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
1469 \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.

```
1470 \def\bbl@switch@sh#1#2{%
1471   \ifx#2\@nnil\else
1472     \bbl@ifunset{bbl@active@\string#2}%
1473       {\bbl@error
1474         {I can't switch '\string#2' on or off--not a shorthand}%
1475         {This character is not a shorthand. Maybe you made\\%
1476          a typing mistake? I will ignore your instruction.}}%
1477       {\ifcase#1%   off, on, off*
1478         \catcode`#212\relax
1479        \or
1480         \catcode`#2\active
1481         \bbl@ifunset{bbl@shdef@\string#2}%
1482           {}%
1483           {\bbl@withactive{\expandafter\let\expandafter}#2%
1484              \csname bbl@shdef@\string#2\endcsname
1485            \bbl@csarg\let{shdef@\string#2}\relax}%
1486         \ifcase\bbl@activated\or
1487           \bbl@activate{#2}%
1488         \else
1489           \bbl@deactivate{#2}%
1490         \fi
1491        \or
1492         \bbl@ifunset{bbl@shdef@\string#2}%
1493           {\bbl@withactive{\bbl@csarg\let{shdef@\string#2}}#2}%
1494           {}%
1495         \csname bbl@oricat@\string#2\endcsname
1496         \csname bbl@oridef@\string#2\endcsname
1497       \fi}%
1498     \bbl@afterfi\bbl@switch@sh#1%
1499   \fi}
```

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

```
1500 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
1501 \def\bbl@putsh#1{%
1502   \bbl@ifunset{bbl@active@\string#1}%
1503     {\bbl@putsh@i#1\@empty\@nnil}%
1504     {\csname bbl@active@\string#1\endcsname}}
```

94

```
1505 \def\bbl@putsh@i#1#2\@nnil{%
1506   \csname\language@group @sh@\string#1@%
1507     \ifx\@empty#2\else\string#2@\fi\endcsname}
1508 \ifx\bbl@opt@shorthands\@nnil\else
1509   \let\bbl@s@initiate@active@char\initiate@active@char
1510   \def\initiate@active@char#1{%
1511     \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
1512   \let\bbl@s@switch@sh\bbl@switch@sh
1513   \def\bbl@switch@sh#1#2{%
1514     \ifx#2\@nnil\else
1515       \bbl@afterfi
1516       \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
1517     \fi}
1518   \let\bbl@s@activate\bbl@activate
1519   \def\bbl@activate#1{%
1520     \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
1521   \let\bbl@s@deactivate\bbl@deactivate
1522   \def\bbl@deactivate#1{%
1523     \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
1524 \fi
```

You may want to test if a character is a shorthand. Note it does not test whether the shorthand is on or off.

```
1525 \newcommand\ifbabelshorthand[3]{\bbl@ifunset{bbl@active@\string#1}{#3}{#2}}
```

\bbl@prim@s
\bbl@pr@m@s    One of the internal macros that are involved in substituting \prime for each right quote in 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.

```
1526 \def\bbl@prim@s{%
1527   \prime\futurelet\@let@token\bbl@pr@m@s}
1528 \def\bbl@if@primes#1#2{%
1529   \ifx#1\@let@token
1530     \expandafter\@firstoftwo
1531   \else\ifx#2\@let@token
1532     \bbl@afterelse\expandafter\@firstoftwo
1533   \else
1534     \bbl@afterfi\expandafter\@secondoftwo
1535   \fi\fi}
1536 \begingroup
1537   \catcode`\^=7  \catcode`\*=\active  \lccode`\*=`\^
1538   \catcode`\'=12 \catcode`\"=\active  \lccode`\"=`\'
1539   \lowercase{%
1540   \gdef\bbl@pr@m@s{%
1541     \bbl@if@primes"'%
1542       \pr@@@s
1543       {\bbl@if@primes*^\pr@@@t\egroup}}}
1544 \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).

```
1545 \initiate@active@char{~}
1546 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
1547 \bbl@activate{~}
```

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

```
1548 \expandafter\def\csname OT1dqpos\endcsname{127}
1549 \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

```
1550 \ifx\f@encoding\@undefined
1551   \def\f@encoding{OT1}
1552 \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.

```
1553 \bbl@trace{Language attributes}
1554 \newcommand\languageattribute[2]{%
1555   \def\bbl@tempc{#1}%
1556   \bbl@fixname\bbl@tempc
1557   \bbl@iflanguage\bbl@tempc{%
1558     \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.

```
1559       \ifx\bbl@known@attribs\@undefined
1560         \in@false
1561       \else
1562         \bbl@xin@{,\bbl@tempc-##1,}{,\bbl@known@attribs,}%
1563       \fi
1564       \ifin@
1565         \bbl@warning{%
1566           You have more than once selected the attribute '##1'\\%
1567           for language #1. Reported}%
1568       \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.

```
1569         \bbl@exp{%
1570           \\\bbl@add@list\\\bbl@known@attribs{\bbl@tempc-##1}}%
1571         \edef\bbl@tempa{\bbl@tempc-##1}%
1572         \expandafter\bbl@ifknown@ttrib\expandafter{\bbl@tempa}\bbl@attributes%
1573         {\csname\bbl@tempc @attr@##1\endcsname}%
1574         {\@attrerr{\bbl@tempc}{##1}}%
1575     \fi}}}
1576 \@onlypreamble\languageattribute
```

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

```
1577 \newcommand*{\@attrerr}[2]{%
1578   \bbl@error
1579     {The attribute #2 is unknown for language #1.}%
1580     {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}.

```
1581 \def\bbl@declare@ttribute#1#2#3{%
1582   \bbl@xin@{,#2,}{,\BabelModifiers,}%
1583   \ifin@
1584     \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
1585   \fi
1586   \bbl@add@list\bbl@attributes{#1-#2}%
1587   \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.

```
1588 \def\bbl@ifattributeset#1#2#3#4{%
1589   \ifx\bbl@known@attribs\@undefined
1590     \in@false
1591   \else
1592     \bbl@xin@{,#1-#2,}{,\bbl@known@attribs,}%
1593   \fi
1594   \ifin@
1595     \bbl@afterelse#3%
1596   \else
1597     \bbl@afterfi#4%
1598   \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.

```
1599 \def\bbl@ifknown@ttrib#1#2{%
1600   \let\bbl@tempa\@secondoftwo
1601   \bbl@loopx\bbl@tempb{#2}{%
1602     \expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{,#1,}%
1603     \ifin@
1604       \let\bbl@tempa\@firstoftwo
1605     \else
1606     \fi}%
1607   \bbl@tempa}
```

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

```
1608 \def\bbl@clear@ttribs{%
1609   \ifx\bbl@attributes\@undefined\else
1610     \bbl@loopx\bbl@tempa{\bbl@attributes}{%
1611       \expandafter\bbl@clear@ttrib\bbl@tempa.
1612       }%
1613     \let\bbl@attributes\@undefined
1614   \fi}
1615 \def\bbl@clear@ttrib#1-#2.{%
1616   \expandafter\let\csname#1@attr@#2\endcsname\@undefined}
1617 \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
\babel@beginsave
The initialization of a new save cycle: reset the counter to zero.

```
1618 \bbl@trace{Macros for saving definitions}
1619 \def\babel@beginsave{\babel@savecnt\z@}
```

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

```
1620 \newcount\babel@savecnt
1621 \babel@beginsave
```

\babel@save
\babel@savevariable

The macro `\babel@save⟨csname⟩` saves the current meaning of the control sequence ⟨*csname*⟩ to `\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 `\babel@savevariable⟨variable⟩` saves the value of the variable. ⟨*variable*⟩ can be anything allowed after the `\the` primitive.

```
1622 \def\babel@save#1{%
1623   \expandafter\let\csname babel@\number\babel@savecnt\endcsname#1\relax
1624   \toks@\expandafter{\originalTeX\let#1=}%
1625   \bbl@exp{%
1626     \def\\\originalTeX{\the\toks@\<babel@\number\babel@savecnt>\relax}}%
1627   \advance\babel@savecnt\@ne}
1628 \def\babel@savevariable#1{%
1629   \toks@\expandafter{\originalTeX #1=}%
1630   \bbl@exp{\def\\\originalTeX{\the\toks@\the#1\relax}}}
```

\bbl@frenchspacing
\bbl@nonfrenchspacing

Some languages need to have `\frenchspacing` in effect. Others don't want that. The command `\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.

```
1631 \def\bbl@frenchspacing{%
1632   \ifnum\the\sfcode`\.=\@m
1633     \let\bbl@nonfrenchspacing\relax
1634   \else
1635     \frenchspacing
1636     \let\bbl@nonfrenchspacing\nonfrenchspacing
1637   \fi}
1638 \let\bbl@nonfrenchspacing\nonfrenchspacing
1639 \let\bbl@elt\relax
1640 \edef\bbl@fs@chars{%
1641   \bbl@elt{\string.}\@m{3000}\bbl@elt{\string?}\@m{3000}%
1642   \bbl@elt{\string!}\@m{3000}\bbl@elt{\string:}\@m{2000}%
1643   \bbl@elt{\string;}\@m{1500}\bbl@elt{\string,}\@m{1250}}
1644 \def\bbl@pre@fs{%
1645   \def\bbl@elt##1##2##3{\sfcode`##1=\the\sfcode`##1\relax}%
1646   \edef\bbl@save@sfcodes{\bbl@fs@chars}}%
1647 \def\bbl@post@fs{%
1648   \bbl@save@sfcodes
1649   \edef\bbl@tempa{\bbl@cl{frspc}}%
1650   \edef\bbl@tempa{\expandafter\@car\bbl@tempa\@nil}%
1651   \if u\bbl@tempa          % do nothing
1652   \else\if n\bbl@tempa     % non french
1653     \def\bbl@elt##1##2##3{%
1654       \ifnum\sfcode`##1=##2\relax
1655         \babel@savevariable{\sfcode`##1}%
1656         \sfcode`##1=##3\relax
1657       \fi}%
1658     \bbl@fs@chars
1659   \else\if y\bbl@tempa     % french
1660     \def\bbl@elt##1##2##3{%
1661       \ifnum\sfcode`##1=##3\relax
1662         \babel@savevariable{\sfcode`##1}%
1663         \sfcode`##1=##2\relax
1664       \fi}%
1665     \bbl@fs@chars
1666   \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

---

[32] `\originalTeX` has to be expandable, i. e. you shouldn't let it to `\relax`.

actual macro.

```
1667 \bbl@trace{Short tags}
1668 \def\babeltags#1{%
1669   \edef\bbl@tempa{\zap@space#1 \@empty}%
1670   \def\bbl@tempb##1=##2\@@{%
1671     \edef\bbl@tempc{%
1672       \noexpand\newcommand
1673       \expandafter\noexpand\csname ##1\endcsname{%
1674         \noexpand\protect
1675         \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}%
1676       \noexpand\newcommand
1677       \expandafter\noexpand\csname text##1\endcsname{%
1678         \noexpand\foreignlanguage{##2}}}%
1679     \bbl@tempc}%
1680   \bbl@for\bbl@tempa\bbl@tempa{%
1681     \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.

```
1682 \bbl@trace{Hyphens}
1683 \@onlypreamble\babelhyphenation
1684 \AtEndOfPackage{%
1685   \newcommand\babelhyphenation[2][\@empty]{%
1686     \ifx\bbl@hyphenation@\relax
1687       \let\bbl@hyphenation@\@empty
1688     \fi
1689     \ifx\bbl@hyphlist\@empty\else
1690       \bbl@warning{%
1691         You must not intermingle \string\selectlanguage\space and\\%
1692         \string\babelhyphenation\space or some exceptions will not\\%
1693         be taken into account. Reported}%
1694     \fi
1695     \ifx\@empty#1%
1696       \protected@edef\bbl@hyphenation@{\bbl@hyphenation@\space#2}%
1697     \else
1698       \bbl@vforeach{#1}{%
1699         \def\bbl@tempa{##1}%
1700         \bbl@fixname\bbl@tempa
1701         \bbl@iflanguage\bbl@tempa{%
1702           \bbl@csarg\protected@edef{hyphenation@\bbl@tempa}{%
1703             \bbl@ifunset{bbl@hyphenation@\bbl@tempa}%
1704               {}%
1705               {\csname bbl@hyphenation@\bbl@tempa\endcsname\space}%
1706             #2}}}%
1707     \fi}}
```

\bbl@allowhyphens    This macro makes hyphenation possible. Basically its definition is nothing more than `\nobreak`
`\hskip 0pt plus 0pt`[33].

```
1708 \def\bbl@allowhyphens{\ifvmode\else\nobreak\hskip\z@skip\fi}
1709 \def\bbl@t@one{T1}
1710 \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`.

```
1711 \newcommand\babelnullhyphen{\char\hyphenchar\font}
1712 \def\babelhyphen{\active@prefix\babelhyphen\bbl@hyphen}
```

---

[33] TeX begins and ends a word for hyphenation at a glue node. The penalty prevents a linebreak at this glue node.

```
1713 \def\bbl@hyphen{%
1714   \@ifstar{\bbl@hyphen@i @}{\bbl@hyphen@i\@empty}}
1715 \def\bbl@hyphen@i#1#2{%
1716   \bbl@ifunset{bbl@hy@#1#2\@empty}%
1717     {\csname bbl@#1usehyphen\endcsname{\discretionary{#2}{}{#2}}}%
1718     {\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.

```
1719 \def\bbl@usehyphen#1{%
1720   \leavevmode
1721   \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
1722   \nobreak\hskip\z@skip}
1723 \def\bbl@@usehyphen#1{%
1724   \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}
```

The following macro inserts the hyphen char.

```
1725 \def\bbl@hyphenchar{%
1726   \ifnum\hyphenchar\font=\m@ne
1727     \babelnullhyphen
1728   \else
1729     \char\hyphenchar\font
1730   \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.

```
1731 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}{}}}
1732 \def\bbl@hy@@soft{\bbl@@usehyphen{\discretionary{\bbl@hyphenchar}{}{}}}
1733 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
1734 \def\bbl@hy@@hard{\bbl@@usehyphen\bbl@hyphenchar}
1735 \def\bbl@hy@nobreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}}
1736 \def\bbl@hy@@nobreak{\mbox{\bbl@hyphenchar}}
1737 \def\bbl@hy@repeat{%
1738   \bbl@usehyphen{%
1739     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
1740 \def\bbl@hy@@repeat{%
1741   \bbl@@usehyphen{%
1742     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
1743 \def\bbl@hy@empty{\hskip\z@skip}
1744 \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.

```
1745 \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 couple of tools. The first one makes global a local variable. This is not the best solution, but it works.

```
1746 \bbl@trace{Multiencoding strings}
1747 \def\bbl@toglobal#1{\global\let#1#1}
1748 \def\bbl@recatcode#1{% TODO. Used only once?
1749   \@tempcnta="7F
1750   \def\bbl@tempa{%
```

100

```
1751    \ifnum\@tempcnta>"FF\else
1752      \catcode\@tempcnta=#1\relax
1753      \advance\@tempcnta\@ne
1754      \expandafter\bbl@tempa
1755    \fi}%
1756  \bbl@tempa}
```

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).

```
1757  \@ifpackagewith{babel}{nocase}%
1758    {\let\bbl@patchuclc\relax}%
1759    {\def\bbl@patchuclc{%
1760      \global\let\bbl@patchuclc\relax
1761      \g@addto@macro\@uclclist{\reserved@b{\reserved@b\bbl@uclc}}%
1762      \gdef\bbl@uclc##1{%
1763        \let\bbl@encoded\bbl@encoded@uclc
1764        \bbl@ifunset{\languagename @bbl@uclc}% and resumes it
1765          {##1}%
1766          {\let\bbl@tempa##1\relax % Used by LANG@bbl@uclc
1767            \csname\languagename @bbl@uclc\endcsname}%
1768        {\bbl@tolower\@empty}{\bbl@toupper\@empty}}%
1769      \gdef\bbl@tolower{\csname\languagename @bbl@lc\endcsname}%
1770      \gdef\bbl@toupper{\csname\languagename @bbl@uc\endcsname}}}
```

```
1771  ⟨⟨*More package options⟩⟩ ≡
1772  \DeclareOption{nocase}{}
1773  ⟨⟨/More package options⟩⟩
```

The following package options control the behavior of \SetString.

```
1774  ⟨⟨*More package options⟩⟩ ≡
1775  \let\bbl@opt@strings\@nnil % accept strings=value
1776  \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
1777  \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
1778  \def\BabelStringsDefault{generic}
1779  ⟨⟨/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.

```
1780  \@onlypreamble\StartBabelCommands
1781  \def\StartBabelCommands{%
1782    \begingroup
1783    \bbl@recatcode{11}%
1784    ⟨⟨Macros local to BabelCommands⟩⟩
1785    \def\bbl@provstring##1##2{%
1786      \providecommand##1{##2}%
1787      \bbl@toglobal##1}%
1788    \global\let\bbl@scafter\@empty
1789    \let\StartBabelCommands\bbl@startcmds
1790    \ifx\BabelLanguages\relax
1791      \let\BabelLanguages\CurrentOption
1792    \fi
1793    \begingroup
1794    \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
```

```
1795    \StartBabelCommands}
1796 \def\bbl@startcmds{%
1797    \ifx\bbl@screset\@nnil\else
1798      \bbl@usehooks{stopcommands}{}%
1799    \fi
1800    \endgroup
1801    \begingroup
1802    \@ifstar
1803      {\ifx\bbl@opt@strings\@nnil
1804         \let\bbl@opt@strings\BabelStringsDefault
1805       \fi
1806       \bbl@startcmds@i}%
1807      \bbl@startcmds@i}
1808 \def\bbl@startcmds@i#1#2{%
1809    \edef\bbl@L{\zap@space#1 \@empty}%
1810    \edef\bbl@G{\zap@space#2 \@empty}%
1811    \bbl@startcmds@ii}
1812 \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.

```
1813 \newcommand\bbl@startcmds@ii[1][\@empty]{%
1814    \let\SetString\@gobbletwo
1815    \let\bbl@stringdef\@gobbletwo
1816    \let\AfterBabelCommands\@gobble
1817    \ifx\@empty#1%
1818      \def\bbl@sc@label{generic}%
1819      \def\bbl@encstring##1##2{%
1820        \ProvideTextCommandDefault##1{##2}%
1821        \bbl@toglobal##1%
1822        \expandafter\bbl@toglobal\csname\string?\string##1\endcsname}%
1823      \let\bbl@sctest\in@true
1824    \else
1825      \let\bbl@sc@charset\space % <- zapped below
1826      \let\bbl@sc@fontenc\space % <-    "        "
1827      \def\bbl@tempa##1=##2\@nil{%
1828        \bbl@csarg\edef{sc@\zap@space##1 \@empty}{##2 }}%
1829      \bbl@vforeach{label=#1}{\bbl@tempa##1\@nil}%
1830      \def\bbl@tempa##1 ##2{% space -> comma
1831        ##1%
1832        \ifx\@empty##2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa##2\fi}%
1833      \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%
1834      \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%
1835      \edef\bbl@sc@charset{\expandafter\zap@space\bbl@sc@charset\@empty}%
1836      \def\bbl@encstring##1##2{%
1837        \bbl@foreach\bbl@sc@fontenc{%
1838          \bbl@ifunset{T@####1}%
1839            {}%
1840            {\ProvideTextCommand##1{####1}{##2}%
1841             \bbl@toglobal##1%
1842             \expandafter
1843             \bbl@toglobal\csname####1\string##1\endcsname}}}%
1844      \def\bbl@sctest{%
1845        \bbl@xin@{,\bbl@opt@strings,}{,\bbl@sc@label,\bbl@sc@fontenc,}}%
1846    \fi
1847    \ifx\bbl@opt@strings\@nnil        % ie, no strings key -> defaults
```

```
1848    \else\ifx\bbl@opt@strings\relax    % ie, strings=encoded
1849      \let\AfterBabelCommands\bbl@aftercmds
1850      \let\SetString\bbl@setstring
1851      \let\bbl@stringdef\bbl@encstring
1852    \else        % ie, strings=value
1853    \bbl@sctest
1854    \ifin@
1855      \let\AfterBabelCommands\bbl@aftercmds
1856      \let\SetString\bbl@setstring
1857      \let\bbl@stringdef\bbl@provstring
1858    \fi\fi\fi
1859    \bbl@scswitch
1860    \ifx\bbl@G\@empty
1861      \def\SetString##1##2{%
1862        \bbl@error{Missing group for string \string##1}%
1863          {You must assign strings to some category, typically\\%
1864            captions or extras, but you set none}}%
1865    \fi
1866    \ifx\@empty#1%
1867      \bbl@usehooks{defaultcommands}{}%
1868    \else
1869      \@expandtwoargs
1870      \bbl@usehooks{encodedcommands}{{\bbl@sc@charset}{\bbl@sc@fontenc}}%
1871    \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) .

```
1872 \def\bbl@forlang#1#2{%
1873   \bbl@for#1\bbl@L{%
1874     \bbl@xin@{,#1,}{,\BabelLanguages,}%
1875     \ifin@#2\relax\fi}}
1876 \def\bbl@scswitch{%
1877   \bbl@forlang\bbl@tempa{%
1878     \ifx\bbl@G\@empty\else
1879       \ifx\SetString\@gobbletwo\else
1880         \edef\bbl@GL{\bbl@G\bbl@tempa}%
1881         \bbl@xin@{,\bbl@GL,}{,\bbl@screset,}%
1882         \ifin@\else
1883           \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
1884           \xdef\bbl@screset{\bbl@screset,\bbl@GL}%
1885         \fi
1886       \fi
1887     \fi}}
1888 \AtEndOfPackage{%
1889   \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{}{#2}}}%
1890   \let\bbl@scswitch\relax}
1891 \@onlypreamble\EndBabelCommands
1892 \def\EndBabelCommands{%
1893   \bbl@usehooks{stopcommands}{}%
1894   \endgroup
1895   \endgroup
1896   \bbl@scafter}
1897 \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.

```
1898 \def\bbl@setstring#1#2{% eg, \prefacename{<string>}
1899   \bbl@forlang\bbl@tempa{%
1900     \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
1901     \bbl@ifunset{\bbl@LC}% eg, \germanchaptername
1902       {\bbl@exp{%
1903         \global\\\bbl@add\<\bbl@G\bbl@tempa>{\\\bbl@scset\\#1\<\bbl@LC>}}}%
1904       {}%
1905     \def\BabelString{#2}%
1906     \bbl@usehooks{stringprocess}{}%
1907     \expandafter\bbl@stringdef
1908       \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.

```
1909 \ifx\bbl@opt@strings\relax
1910   \def\bbl@scset#1#2{\def#1{\bbl@encoded#2}}
1911   \bbl@patchuclc
1912   \let\bbl@encoded\relax
1913   \def\bbl@encoded@uclc#1{%
1914     \@inmathwarn#1%
1915     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
1916       \expandafter\ifx\csname ?\string#1\endcsname\relax
1917         \TextSymbolUnavailable#1%
1918       \else
1919         \csname ?\string#1\endcsname
1920       \fi
1921     \else
1922       \csname\cf@encoding\string#1\endcsname
1923     \fi}
1924 \else
1925   \def\bbl@scset#1#2{\def#1{#2}}
1926 \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.

```
1927 ⟨⟨*Macros local to BabelCommands⟩⟩ ≡
1928 \def\SetStringLoop##1##2{%
1929   \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
1930   \count@\z@
1931   \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
1932     \advance\count@\@ne
1933     \toks@\expandafter{\bbl@tempa}%
1934     \bbl@exp{%
1935       \\\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
1936       \count@=\the\count@\relax}}}
1937 ⟨⟨/Macros local to BabelCommands⟩⟩
```

**Delaying code**    Now the definition of \AfterBabelCommands when it is activated.

```
1938 \def\bbl@aftercmds#1{%
1939   \toks@\expandafter{\bbl@scafter#1}%
1940   \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.

```
1941 ⟨⟨*Macros local to BabelCommands⟩⟩ ≡
```

```
1942   \newcommand\SetCase[3][]{%
1943     \bbl@patchuclc
1944     \bbl@forlang\bbl@tempa{%
1945       \expandafter\bbl@encstring
1946         \csname\bbl@tempa @bbl@uclc\endcsname{\bbl@tempa##1}%
1947       \expandafter\bbl@encstring
1948         \csname\bbl@tempa @bbl@uc\endcsname{##2}%
1949       \expandafter\bbl@encstring
1950         \csname\bbl@tempa @bbl@lc\endcsname{##3}}}%
1951 ⟨⟨/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.

```
1952 ⟨⟨*Macros local to BabelCommands⟩⟩ ≡
1953   \newcommand\SetHyphenMap[1]{%
1954     \bbl@forlang\bbl@tempa{%
1955       \expandafter\bbl@stringdef
1956         \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
1957 ⟨⟨/Macros local to BabelCommands⟩⟩
```

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

```
1958 \newcommand\BabelLower[2]{% one to one.
1959   \ifnum\lccode#1=#2\else
1960     \babel@savevariable{\lccode#1}%
1961     \lccode#1=#2\relax
1962   \fi}
1963 \newcommand\BabelLowerMM[4]{% many-to-many
1964   \@tempcnta=#1\relax
1965   \@tempcntb=#4\relax
1966   \def\bbl@tempa{%
1967     \ifnum\@tempcnta>#2\else
1968       \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
1969       \advance\@tempcnta#3\relax
1970       \advance\@tempcntb#3\relax
1971       \expandafter\bbl@tempa
1972     \fi}%
1973   \bbl@tempa}
1974 \newcommand\BabelLowerMO[4]{% many-to-one
1975   \@tempcnta=#1\relax
1976   \def\bbl@tempa{%
1977     \ifnum\@tempcnta>#2\else
1978       \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
1979       \advance\@tempcnta#3
1980       \expandafter\bbl@tempa
1981     \fi}%
1982   \bbl@tempa}
```

The following package options control the behavior of hyphenation mapping.

```
1983 ⟨⟨*More package options⟩⟩ ≡
1984 \DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\z@}
1985 \DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap\@ne}
1986 \DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\tw@}
1987 \DeclareOption{hyphenmap=other}{\chardef\bbl@opt@hyphenmap\thr@@}
1988 \DeclareOption{hyphenmap=other*}{\chardef\bbl@opt@hyphenmap4\relax}
1989 ⟨⟨/More package options⟩⟩
```

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

```
1990 \AtEndOfPackage{%
1991   \ifx\bbl@opt@hyphenmap\@undefined
1992     \bbl@xin@{,}{\bbl@language@opts}%
1993     \chardef\bbl@opt@hyphenmap\ifin@4\else\@ne\fi
1994   \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.

```
1995 \newcommand\setlocalecaption{%  TODO. Catch typos. What about ensure?
1996   \@ifstar\bbl@setcaption@s\bbl@setcaption@x}
1997 \def\bbl@setcaption@x#1#2#3{%  language caption-name string
1998   \bbl@trim@def\bbl@tempa{#2}%
1999   \bbl@xin@{.template}{\bbl@tempa}%
2000   \ifin@
2001     \bbl@ini@captions@template{#3}{#1}%
2002   \else
2003     \edef\bbl@tempd{%
2004       \expandafter\expandafter\expandafter
2005       \strip@prefix\expandafter\meaning\csname captions#1\endcsname}%
2006     \bbl@xin@
2007       {\expandafter\string\csname #2name\endcsname}%
2008       {\bbl@tempd}%
2009     \ifin@ % Renew caption
2010       \bbl@xin@{\string\bbl@scset}{\bbl@tempd}%
2011       \ifin@
2012         \bbl@exp{%
2013           \\\bbl@ifsamestring{\bbl@tempa}{\languagename}%
2014             {\\\bbl@scset\<#2name>\<#1#2name>}%
2015             {}}%
2016       \else % Old way converts to new way
2017         \bbl@ifunset{#1#2name}%
2018           {\bbl@exp{%
2019             \\\bbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}%
2020             \\\bbl@ifsamestring{\bbl@tempa}{\languagename}%
2021               {\def\<#2name>{\<#1#2name>}}%
2022               {}}}%
2023           {}%
2024       \fi
2025     \else
2026       \bbl@xin@{\string\bbl@scset}{\bbl@tempd}% New
2027       \ifin@ % New way
2028         \bbl@exp{%
2029           \\\bbl@add\<captions#1>{\\\bbl@scset\<#2name>\<#1#2name>}%
2030           \\\bbl@ifsamestring{\bbl@tempa}{\languagename}%
2031             {\\\bbl@scset\<#2name>\<#1#2name>}%
2032             {}}%
2033       \else  % Old way, but defined in the new way
2034         \bbl@exp{%
2035           \\\bbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}%
2036           \\\bbl@ifsamestring{\bbl@tempa}{\languagename}%
2037             {\def\<#2name>{\<#1#2name>}}%
2038             {}}%
2039       \fi%
2040     \fi
2041     \@namedef{#1#2name}{#3}%
2042     \toks@\expandafter{\bbl@captionslist}%
2043     \bbl@exp{\\\in@{\<#2name>}{\the\toks@}}%
2044     \ifin@\else
2045       \bbl@exp{\\\bbl@add\\\bbl@captionslist{\<#2name>}}%
2046       \bbl@toglobal\bbl@captionslist
2047     \fi
2048   \fi}
2049 % \def\bbl@setcaption@s#1#2#3{}  % TODO. Not yet implemented
```

## 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.

```
2050 \bbl@trace{Macros related to glyphs}
2051 \def\set@low@box#1{\setbox\tw@\hbox{,}\setbox\z@\hbox{#1}%
2052    \dimen\z@\ht\z@ \advance\dimen\z@ -\ht\tw@%
2053    \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.

```
2054 \def\save@sf@q#1{\leavevmode
2055   \begingroup
2056     \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
2057   \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.

```
2058 \ProvideTextCommand{\quotedblbase}{OT1}{%
2059   \save@sf@q{\set@low@box{\textquotedblright\/}%
2060     \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.

```
2061 \ProvideTextCommandDefault{\quotedblbase}{%
2062   \UseTextSymbol{OT1}{\quotedblbase}}
```

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

```
2063 \ProvideTextCommand{\quotesinglbase}{OT1}{%
2064   \save@sf@q{\set@low@box{\textquoteright\/}%
2065     \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.

```
2066 \ProvideTextCommandDefault{\quotesinglbase}{%
2067   \UseTextSymbol{OT1}{\quotesinglbase}}
```

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

```
2068 \ProvideTextCommand{\guillemetleft}{OT1}{%
2069   \ifmmode
2070     \ll
2071   \else
2072     \save@sf@q{\nobreak
2073       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2074   \fi}
2075 \ProvideTextCommand{\guillemetright}{OT1}{%
2076   \ifmmode
2077     \gg
2078   \else
2079     \save@sf@q{\nobreak
2080       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2081   \fi}
2082 \ProvideTextCommand{\guillemotleft}{OT1}{%
2083   \ifmmode
2084     \ll
2085   \else
```

```
2086     \save@sf@q{\nobreak
2087       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2088   \fi}
2089 \ProvideTextCommand{\guillemotright}{OT1}{%
2090   \ifmmode
2091     \gg
2092   \else
2093     \save@sf@q{\nobreak
2094       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2095   \fi}
```

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

```
2096 \ProvideTextCommandDefault{\guillemetleft}{%
2097   \UseTextSymbol{OT1}{\guillemetleft}}
2098 \ProvideTextCommandDefault{\guillemetright}{%
2099   \UseTextSymbol{OT1}{\guillemetright}}
2100 \ProvideTextCommandDefault{\guillemotleft}{%
2101   \UseTextSymbol{OT1}{\guillemotleft}}
2102 \ProvideTextCommandDefault{\guillemotright}{%
2103   \UseTextSymbol{OT1}{\guillemotright}}
```

`\guilsinglleft`
`\guilsinglright` The single guillemets are not available in OT1 encoding. They are faked.

```
2104 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2105   \ifmmode
2106     <%
2107   \else
2108     \save@sf@q{\nobreak
2109       \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
2110   \fi}
2111 \ProvideTextCommand{\guilsinglright}{OT1}{%
2112   \ifmmode
2113     >%
2114   \else
2115     \save@sf@q{\nobreak
2116       \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
2117   \fi}
```

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

```
2118 \ProvideTextCommandDefault{\guilsinglleft}{%
2119   \UseTextSymbol{OT1}{\guilsinglleft}}
2120 \ProvideTextCommandDefault{\guilsinglright}{%
2121   \UseTextSymbol{OT1}{\guilsinglright}}
```

### 7.12.2 Letters

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

```
2122 \DeclareTextCommand{\ij}{OT1}{%
2123   i\kern-0.02em\bbl@allowhyphens j}
2124 \DeclareTextCommand{\IJ}{OT1}{%
2125   I\kern-0.02em\bbl@allowhyphens J}
2126 \DeclareTextCommand{\ij}{T1}{\char188}
2127 \DeclareTextCommand{\IJ}{T1}{\char156}
```

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

```
2128 \ProvideTextCommandDefault{\ij}{%
2129   \UseTextSymbol{OT1}{\ij}}
2130 \ProvideTextCommandDefault{\IJ}{%
2131   \UseTextSymbol{OT1}{\IJ}}
```

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

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

```
2132 \def\crrtic@{\hrule height0.1ex width0.3em}
2133 \def\crttic@{\hrule height0.1ex width0.33em}
2134 \def\ddj@{%
2135    \setbox0\hbox{d}\dimen@=\ht0
2136    \advance\dimen@1ex
2137    \dimen@.45\dimen@
2138    \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2139    \advance\dimen@ii.5ex
2140    \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2141 \def\DDJ@{%
2142    \setbox0\hbox{D}\dimen@=.55\ht0
2143    \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2144    \advance\dimen@ii.15ex %              correction for the dash position
2145    \advance\dimen@ii-.15\fontdimen7\font %    correction for cmtt font
2146    \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
2147    \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
2148 %
2149 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2150 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}
```

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

```
2151 \ProvideTextCommandDefault{\dj}{%
2152    \UseTextSymbol{OT1}{\dj}}
2153 \ProvideTextCommandDefault{\DJ}{%
2154    \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.

```
2155 \DeclareTextCommand{\SS}{OT1}{SS}
2156 \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
```
2157 \ProvideTextCommandDefault{\glq}{%
2158    \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}
```

The definition of \grq depends on the fontencoding. With T1 encoding no extra kerning is needed.

```
2159 \ProvideTextCommand{\grq}{T1}{%
2160    \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}
2161 \ProvideTextCommand{\grq}{TU}{%
2162    \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
2163 \ProvideTextCommand{\grq}{OT1}{%
2164    \save@sf@q{\kern-.0125em
2165       \textormath{\textquoteleft}{\mbox{\textquoteleft}}%
2166       \kern.07em\relax}}
2167 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}\grq}
```

\glqq   The 'german' double quotes.
\grqq
```
2168 \ProvideTextCommandDefault{\glqq}{%
2169    \textormath{\quotedblbase}{\mbox{\quotedblbase}}}
```

The definition of \grqq depends on the fontencoding. With T1 encoding no extra kerning is needed.

```
2170 \ProvideTextCommand{\grqq}{T1}{%
2171    \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2172 \ProvideTextCommand{\grqq}{TU}{%
2173    \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
```

```
2174 \ProvideTextCommand{\grqq}{OT1}{%
2175   \save@sf@q{\kern-.07em
2176     \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}%
2177     \kern.07em\relax}}
2178 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}
```

\flq  The 'french' single guillemets.
\frq
```
2179 \ProvideTextCommandDefault{\flq}{%
2180   \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
2181 \ProvideTextCommandDefault{\frq}{%
2182   \textormath{\guilsinglright}{\mbox{\guilsinglright}}}
```

\flqq  The 'french' double guillemets.
\frqq
```
2183 \ProvideTextCommandDefault{\flqq}{%
2184   \textormath{\guillemetleft}{\mbox{\guillemetleft}}}
2185 \ProvideTextCommandDefault{\frqq}{%
2186   \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).
```
2187 \def\umlauthigh{%
2188   \def\bbl@umlauta##1{\leavevmode\bgroup%
2189     \expandafter\accent\csname\f@encoding dqpos\endcsname
2190     ##1\bbl@allowhyphens\egroup}%
2191   \let\bbl@umlaute\bbl@umlauta}
2192 \def\umlautlow{%
2193   \def\bbl@umlauta{\protect\lower@umlaut}}
2194 \def\umlautelow{%
2195   \def\bbl@umlaute{\protect\lower@umlaut}}
2196 \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.
```
2197 \expandafter\ifx\csname U@D\endcsname\relax
2198   \csname newdimen\endcsname\U@D
2199 \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.
```
2200 \def\lower@umlaut#1{%
2201   \leavevmode\bgroup
2202     \U@D 1ex%
2203     {\setbox\z@\hbox{%
2204       \expandafter\char\csname\f@encoding dqpos\endcsname}%
2205       \dimen@ -.45ex\advance\dimen@\ht\z@
2206       \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2207     \expandafter\accent\csname\f@encoding dqpos\endcsname
2208     \fontdimen5\font\U@D #1%
2209   \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).

```
2210 \AtBeginDocument{%
2211   \DeclareTextCompositeCommand{\"}{OT1}{a}{\bbl@umlauta{a}}%
2212   \DeclareTextCompositeCommand{\"}{OT1}{e}{\bbl@umlaute{e}}%
2213   \DeclareTextCompositeCommand{\"}{OT1}{i}{\bbl@umlaute{\i}}%
2214   \DeclareTextCompositeCommand{\"}{OT1}{\i}{\bbl@umlaute{\i}}%
2215   \DeclareTextCompositeCommand{\"}{OT1}{o}{\bbl@umlauta{o}}%
2216   \DeclareTextCompositeCommand{\"}{OT1}{u}{\bbl@umlauta{u}}%
2217   \DeclareTextCompositeCommand{\"}{OT1}{A}{\bbl@umlauta{A}}%
2218   \DeclareTextCompositeCommand{\"}{OT1}{E}{\bbl@umlaute{E}}%
2219   \DeclareTextCompositeCommand{\"}{OT1}{I}{\bbl@umlaute{I}}%
2220   \DeclareTextCompositeCommand{\"}{OT1}{O}{\bbl@umlauta{O}}%
2221   \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.

```
2222 \ifx\l@english\@undefined
2223   \chardef\l@english\z@
2224 \fi
2225 % The following is used to cancel rules in ini files (see Amharic).
2226 \ifx\l@unhyphenated\@undefined
2227   \newlanguage\l@unhyphenated
2228 \fi
```

## 7.13 Layout

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

```
2229 \bbl@trace{Bidi layout}
2230 \providecommand\IfBabelLayout[3]{#3}%
2231 \newcommand\BabelPatchSection[1]{%
2232   \@ifundefined{#1}{}{%
2233     \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
2234     \@namedef{#1}{%
2235       \@ifstar{\bbl@presec@s{#1}}%
2236               {\@dblarg{\bbl@presec@x{#1}}}}}}
2237 \def\bbl@presec@x#1[#2]#3{%
2238   \bbl@exp{%
2239     \\\select@language@x{\bbl@main@language}%
2240     \\\bbl@cs{sspre@#1}%
2241     \\\bbl@cs{ss@#1}%
2242       [\\\foreignlanguage{\languagename}{\unexpanded{#2}}]%
2243       {\\\foreignlanguage{\languagename}{\unexpanded{#3}}}%
2244     \\\select@language@x{\languagename}}}
2245 \def\bbl@presec@s#1#2{%
2246   \bbl@exp{%
2247     \\\select@language@x{\bbl@main@language}%
2248     \\\bbl@cs{sspre@#1}%
2249     \\\bbl@cs{ss@#1}*%
2250       {\\\foreignlanguage{\languagename}{\unexpanded{#2}}}%
2251     \\\select@language@x{\languagename}}}
2252 \IfBabelLayout{sectioning}%
2253   {\BabelPatchSection{part}%
2254    \BabelPatchSection{chapter}%
2255    \BabelPatchSection{section}%
2256    \BabelPatchSection{subsection}%
2257    \BabelPatchSection{subsubsection}%
2258    \BabelPatchSection{paragraph}%
```

```
2259    \BabelPatchSection{subparagraph}%
2260    \def\babel@toc#1{%
2261      \select@language@x{\bbl@main@language}}}{}
2262 \IfBabelLayout{captions}%
2263   {\BabelPatchSection{caption}}{}
```

## 7.14  Load engine specific macros

```
2264 \bbl@trace{Input engine specific macros}
2265 \ifcase\bbl@engine
2266   \input txtbabel.def
2267 \or
2268   \input luababel.def
2269 \or
2270   \input xebabel.def
2271 \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.

```
2272 \bbl@trace{Creating languages and reading ini files}
2273 \let\bbl@extend@ini\@gobble
2274 \newcommand\babelprovide[2][]{%
2275   \let\bbl@savelangname\languagename
2276   \edef\bbl@savelocaleid{\the\localeid}%
2277   % Set name and locale id
2278   \edef\languagename{#2}%
2279   \bbl@id@assign
2280   % Initialize keys
2281   \let\bbl@KVP@captions\@nil
2282   \let\bbl@KVP@date\@nil
2283   \let\bbl@KVP@import\@nil
2284   \let\bbl@KVP@main\@nil
2285   \let\bbl@KVP@script\@nil
2286   \let\bbl@KVP@language\@nil
2287   \let\bbl@KVP@hyphenrules\@nil
2288   \let\bbl@KVP@linebreaking\@nil
2289   \let\bbl@KVP@justification\@nil
2290   \let\bbl@KVP@mapfont\@nil
2291   \let\bbl@KVP@maparabic\@nil
2292   \let\bbl@KVP@mapdigits\@nil
2293   \let\bbl@KVP@intraspace\@nil
2294   \let\bbl@KVP@intrapenalty\@nil
2295   \let\bbl@KVP@onchar\@nil
2296   \let\bbl@KVP@transforms\@nil
2297   \global\let\bbl@release@transforms\@empty
2298   \let\bbl@KVP@alph\@nil
2299   \let\bbl@KVP@Alph\@nil
2300   \let\bbl@KVP@labels\@nil
2301   \bbl@csarg\let{KVP@labels*}\@nil
2302   \let\bbl@KVP@calendar\@nil
2303   \let\bbl@calendars\@empty
2304   \global\let\bbl@inidata\@empty
2305   \global\let\bbl@extend@ini\@gobble
2306   \gdef\bbl@key@list{;}%
2307   \bbl@forkv{#1}{%  TODO - error handling
2308     \in@{/}{##1}%
2309     \ifin@
2310       \global\let\bbl@extend@ini\bbl@extend@ini@aux
2311       \bbl@renewinikey##1\@@{##2}%
2312     \else
```

```
2313        \bbl@csarg\def{KVP@##1}{##2}%
2314      \fi}%
2315  \chardef\bbl@howloaded=% 0:none; 1:ldf without ini; 2:ini
2316      \bbl@ifunset{date#2}\z@{\bbl@ifunset{bbl@llevel@#2}\@ne\tw@}%
2317  % == init ==
2318  \ifx\bbl@screset\@undefined
2319      \bbl@ldfinit
2320  \fi
2321  % ==
2322  \let\bbl@lbkflag\relax % \@empty = do setup linebreak
2323  \ifcase\bbl@howloaded
2324      \let\bbl@lbkflag\@empty % new
2325  \else
2326      \ifx\bbl@KVP@hyphenrules\@nil\else
2327          \let\bbl@lbkflag\@empty
2328      \fi
2329      \ifx\bbl@KVP@import\@nil\else
2330          \let\bbl@lbkflag\@empty
2331      \fi
2332  \fi
2333  % == import, captions ==
2334  \ifx\bbl@KVP@import\@nil\else
2335      \bbl@exp{\\\bbl@ifblank{\bbl@KVP@import}}%
2336          {\ifx\bbl@initoload\relax
2337              \begingroup
2338                  \def\BabelBeforeIni##1##2{\gdef\bbl@KVP@import{##1}\endinput}%
2339                  \bbl@input@texini{#2}%
2340              \endgroup
2341          \else
2342              \xdef\bbl@KVP@import{\bbl@initoload}%
2343          \fi}%
2344          {}%
2345  \fi
2346  \ifx\bbl@KVP@captions\@nil
2347      \let\bbl@KVP@captions\bbl@KVP@import
2348  \fi
2349  % ==
2350  \ifx\bbl@KVP@transforms\@nil\else
2351      \bbl@replace\bbl@KVP@transforms{ }{,}%
2352  \fi
2353  % == Load ini ==
2354  \ifcase\bbl@howloaded
2355      \bbl@provide@new{#2}%
2356  \else
2357      \bbl@ifblank{#1}%
2358          {}%  With \bbl@load@basic below
2359          {\bbl@provide@renew{#2}}%
2360  \fi
2361  % Post tasks
2362  % ----------
2363  % == subsequent calls after the first provide for a locale ==
2364  \ifx\bbl@inidata\@empty\else
2365      \bbl@extend@ini{#2}%
2366  \fi
2367  % == ensure captions ==
2368  \ifx\bbl@KVP@captions\@nil\else
2369      \bbl@ifunset{bbl@extracaps@#2}%
2370          {\bbl@exp{\\\babelensure[exclude=\\\today]{#2}}}%
2371          {\bbl@exp{\\\babelensure[exclude=\\\today,
2372                      include=\[bbl@extracaps@#2]]{#2}}%
2373      \bbl@ifunset{bbl@ensure@\languagename}%
2374          {\bbl@exp{%
2375              \\\DeclareRobustCommand\<bbl@ensure@\languagename>[1]{%
```

```
2376          \\\foreignlanguage{\languagename}%
2377            {####1}}}}%
2378       {}%
2379    \bbl@exp{%
2380       \\\bbl@toglobal\<bbl@ensure@\languagename>%
2381       \\\bbl@toglobal\<bbl@ensure@\languagename\space>}%
2382    \fi
2383    % ==
2384    % At this point all parameters are defined if 'import'. Now we
2385    % execute some code depending on them. But what about if nothing was
2386    % imported? We just set the basic parameters, but still loading the
2387    % whole ini file.
2388    \bbl@load@basic{#2}%
2389    % == script, language ==
2390    % Override the values from ini or defines them
2391    \ifx\bbl@KVP@script\@nil\else
2392       \bbl@csarg\edef{sname@#2}{\bbl@KVP@script}%
2393    \fi
2394    \ifx\bbl@KVP@language\@nil\else
2395       \bbl@csarg\edef{lname@#2}{\bbl@KVP@language}%
2396    \fi
2397    \ifcase\bbl@engine\or
2398       \bbl@ifunset{bbl@chrng@\languagename}{}%
2399          {\directlua{
2400             Babel.set_chranges_b('\bbl@cl{sbcp}', '\bbl@cl{chrng}') }}%
2401    \fi
2402     % == onchar ==
2403    \ifx\bbl@KVP@onchar\@nil\else
2404       \bbl@luahyphenate
2405       \bbl@exp{%
2406          \\\AddToHook{env/document/before}{{\\\select@language{#2}{}}}}%
2407       \directlua{
2408          if Babel.locale_mapped == nil then
2409            Babel.locale_mapped = true
2410            Babel.linebreaking.add_before(Babel.locale_map)
2411            Babel.loc_to_scr = {}
2412            Babel.chr_to_loc = Babel.chr_to_loc or {}
2413          end}%
2414       \bbl@xin@{ ids }{ \bbl@KVP@onchar\space}%
2415       \ifin@
2416          \ifx\bbl@starthyphens\@undefined % Needed if no explicit selection
2417            \AddBabelHook{babel-onchar}{beforestart}{{\bbl@starthyphens}}%
2418          \fi
2419          \bbl@exp{\\\bbl@add\\\bbl@starthyphens
2420            {\\\bbl@patterns@lua{\languagename}}}%
2421          % TODO - error/warning if no script
2422          \directlua{
2423            if Babel.script_blocks['\bbl@cl{sbcp}'] then
2424              Babel.loc_to_scr[\the\localeid] =
2425                Babel.script_blocks['\bbl@cl{sbcp}']
2426              Babel.locale_props[\the\localeid].lc = \the\localeid\space
2427              Babel.locale_props[\the\localeid].lg = \the\@nameuse{l@\languagename}\space
2428            end
2429          }%
2430       \fi
2431       \bbl@xin@{ fonts }{ \bbl@KVP@onchar\space}%
2432       \ifin@
2433          \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
2434          \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
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          end}%
2440       \ifx\bbl@mapselect\@undefined  % TODO. almost the same as mapfont
2441         \AtBeginDocument{%
2442           \bbl@patchfont{{\bbl@mapselect}}%
2443           {\selectfont}}%
2444         \def\bbl@mapselect{%
2445           \let\bbl@mapselect\relax
2446           \edef\bbl@prefontid{\fontid\font}}%
2447         \def\bbl@mapdir##1{%
2448           {\def\languagename{##1}%
2449            \let\bbl@ifrestoring\@firstoftwo % To avoid font warning
2450            \bbl@switchfont
2451            \ifnum\fontid\font>\z@ % A hack, for the pgf nullfont hack
2452              \directlua{
2453                Babel.locale_props[\the\csname bbl@id@@##1\endcsname]%
2454                         ['/\bbl@prefontid'] = \fontid\font\space}%
2455            \fi}}%
2456       \fi
2457       \bbl@exp{\\\bbl@add\\\bbl@mapselect{\\\bbl@mapdir{\languagename}}}%
2458     \fi
2459     % TODO - catch non-valid values
2460   \fi
2461 % == mapfont ==
2462 % For bidi texts, to switch the font based on direction
2463 \ifx\bbl@KVP@mapfont\@nil\else
2464   \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}{}%
2465     {\bbl@error{Option '\bbl@KVP@mapfont' unknown for\\%
2466                 mapfont. Use 'direction'.%
2467                {See the manual for details.}}}%
2468   \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
2469   \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
2470   \ifx\bbl@mapselect\@undefined % TODO. See onchar.
2471     \AtBeginDocument{%
2472       \bbl@patchfont{{\bbl@mapselect}}%
2473       {\selectfont}}%
2474     \def\bbl@mapselect{%
2475       \let\bbl@mapselect\relax
2476       \edef\bbl@prefontid{\fontid\font}}%
2477     \def\bbl@mapdir##1{%
2478       {\def\languagename{##1}%
2479        \let\bbl@ifrestoring\@firstoftwo % avoid font warning
2480        \bbl@switchfont
2481        \directlua{Babel.fontmap
2482          [\the\csname bbl@wdir@##1\endcsname]%
2483          [\bbl@prefontid]=\fontid\font}}}%
2484   \fi
2485   \bbl@exp{\\\bbl@add\\\bbl@mapselect{\\\bbl@mapdir{\languagename}}}%
2486 \fi
2487 % == Line breaking: intraspace, intrapenalty ==
2488 % For CJK, East Asian, Southeast Asian, if interspace in ini
2489 \ifx\bbl@KVP@intraspace\@nil\else % We can override the ini or set
2490   \bbl@csarg\edef{intsp@#2}{\bbl@KVP@intraspace}%
2491 \fi
2492 \bbl@provide@intraspace
2493 % == Line breaking: CJK quotes ==
2494 \ifcase\bbl@engine\or
2495   \bbl@xin@{/c}{/\bbl@cl{lnbrk}}%
2496   \ifin@
2497     \bbl@ifunset{bbl@quote@\languagename}{}%
2498       {\directlua{
2499         Babel.locale_props[\the\localeid].cjk_quotes = {}
2500         local cs = 'op'
2501         for c in string.utfvalues(%
```

115

```
2502            [[\csname bbl@quote@\languagename\endcsname]]) do
2503              if Babel.cjk_characters[c].c == 'qu' then
2504                Babel.locale_props[\the\localeid].cjk_quotes[c] = cs
2505              end
2506              cs = ( cs == 'op') and 'cl' or 'op'
2507            end
2508        }}%
2509    \fi
2510  \fi
2511  % == Line breaking: justification ==
2512  \ifx\bbl@KVP@justification\@nil\else
2513    \let\bbl@KVP@linebreaking\bbl@KVP@justification
2514  \fi
2515  \ifx\bbl@KVP@linebreaking\@nil\else
2516    \bbl@xin@{,\bbl@KVP@linebreaking,}{,elongated,kashida,cjk,unhyphenated,}%
2517    \ifin@
2518      \bbl@csarg\xdef
2519        {lnbrk@\languagename}{\expandafter\@car\bbl@KVP@linebreaking\@nil}%
2520    \fi
2521  \fi
2522  \bbl@xin@{/e}{/\bbl@cl{lnbrk}}%
2523  \ifin@\else\bbl@xin@{/k}{/\bbl@cl{lnbrk}}\fi
2524  \ifin@\bbl@arabicjust\fi
2525  % == Line breaking: hyphenate.other.(locale|script) ==
2526  \ifx\bbl@lbkflag\@empty
2527    \bbl@ifunset{bbl@hyotl@\languagename}{}%
2528      {\bbl@csarg\bbl@replace{hyotl@\languagename}{ }{,}%
2529        \bbl@startcommands*{\languagename}{}%
2530          \bbl@csarg\bbl@foreach{hyotl@\languagename}{%
2531            \ifcase\bbl@engine
2532              \ifnum##1<257
2533                \SetHyphenMap{\BabelLower{##1}{##1}}%
2534              \fi
2535            \else
2536              \SetHyphenMap{\BabelLower{##1}{##1}}%
2537            \fi}%
2538        \bbl@endcommands}%
2539    \bbl@ifunset{bbl@hyots@\languagename}{}%
2540      {\bbl@csarg\bbl@replace{hyots@\languagename}{ }{,}%
2541        \bbl@csarg\bbl@foreach{hyots@\languagename}{%
2542          \ifcase\bbl@engine
2543            \ifnum##1<257
2544              \global\lccode##1=##1\relax
2545            \fi
2546          \else
2547            \global\lccode##1=##1\relax
2548          \fi}}%
2549  \fi
2550  % == Counters: maparabic ==
2551  % Native digits, if provided in ini (TeX level, xe and lua)
2552  \ifcase\bbl@engine\else
2553    \bbl@ifunset{bbl@dgnat@\languagename}{}%
2554      {\expandafter\ifx\csname bbl@dgnat@\languagename\endcsname\@empty\else
2555        \expandafter\expandafter\expandafter
2556        \bbl@setdigits\csname bbl@dgnat@\languagename\endcsname
2557        \ifx\bbl@KVP@maparabic\@nil\else
2558          \ifx\bbl@latinarabic\@undefined
2559            \expandafter\let\expandafter\@arabic
2560              \csname bbl@counter@\languagename\endcsname
2561          \else    % ie, if layout=counters, which redefines \@arabic
2562            \expandafter\let\expandafter\bbl@latinarabic
2563              \csname bbl@counter@\languagename\endcsname
2564          \fi
```

```
2565        \fi
2566      \fi}%
2567    \fi
2568  % == Counters: mapdigits ==
2569  % Native digits (lua level).
2570  \ifodd\bbl@engine
2571    \ifx\bbl@KVP@mapdigits\@nil\else
2572      \bbl@ifunset{bbl@dgnat@\languagename}{}%
2573        {\RequirePackage{luatexbase}%
2574         \bbl@activate@preotf
2575         \directlua{
2576           Babel = Babel or {}  %%% -> presets in luababel
2577           Babel.digits_mapped = true
2578           Babel.digits = Babel.digits or {}
2579           Babel.digits[\the\localeid] =
2580             table.pack(string.utfvalue('\bbl@cl{dgnat}'))
2581           if not Babel.numbers then
2582             function Babel.numbers(head)
2583               local LOCALE = Babel.attr_locale
2584               local GLYPH = node.id'glyph'
2585               local inmath = false
2586               for item in node.traverse(head) do
2587                 if not inmath and item.id == GLYPH then
2588                   local temp = node.get_attribute(item, LOCALE)
2589                   if Babel.digits[temp] then
2590                     local chr = item.char
2591                     if chr > 47 and chr < 58 then
2592                       item.char = Babel.digits[temp][chr-47]
2593                     end
2594                   end
2595                 elseif item.id == node.id'math' then
2596                   inmath = (item.subtype == 0)
2597                 end
2598               end
2599               return head
2600             end
2601           end
2602         }}%
2603    \fi
2604  \fi
2605  % == Counters: alph, Alph ==
2606  % What if extras<lang> contains a \babel@save\@alph? It won't be
2607  % restored correctly when exiting the language, so we ignore
2608  % this change with the \bbl@alph@saved trick.
2609  \ifx\bbl@KVP@alph\@nil\else
2610    \bbl@extras@wrap{\\\bbl@alph@saved}%
2611      {\let\bbl@alph@saved\@alph}%
2612      {\let\@alph\bbl@alph@saved
2613       \babel@save\@alph}%
2614    \bbl@exp{%
2615      \\\bbl@add\<extras\languagename>{%
2616        \let\\\@alph\<bbl@cntr@\bbl@KVP@alph @\languagename>}}%
2617  \fi
2618  \ifx\bbl@KVP@Alph\@nil\else
2619    \bbl@extras@wrap{\\\bbl@Alph@saved}%
2620      {\let\bbl@Alph@saved\@Alph}%
2621      {\let\@Alph\bbl@Alph@saved
2622       \babel@save\@Alph}%
2623    \bbl@exp{%
2624      \\\bbl@add\<extras\languagename>{%
2625        \let\\\@Alph\<bbl@cntr@\bbl@KVP@Alph @\languagename>}}%
2626  \fi
2627  % == Calendars ==
```

```
2628    \ifx\bbl@KVP@calendar\@nil\else
2629      \bbl@csarg\let{calpr@\languagename}\bbl@KVP@calendar
2630    \fi
2631    % == require.babel in ini ==
2632    % To load or reaload the babel-*.tex, if require.babel in ini
2633    \ifx\bbl@beforestart\relax\else  % But not in doc aux or body
2634      \bbl@ifunset{bbl@rqtex@\languagename}{}%
2635        {\expandafter\ifx\csname bbl@rqtex@\languagename\endcsname\@empty\else
2636          \let\BabelBeforeIni\@gobbletwo
2637          \chardef\atcatcode=\catcode`\@
2638          \catcode`\@=11\relax
2639          \bbl@input@texini{\bbl@cs{rqtex@\languagename}}%
2640          \catcode`\@=\atcatcode
2641          \let\atcatcode\relax
2642          \global\bbl@csarg\let{rqtex@\languagename}\relax
2643        \fi}%
2644      \bbl@foreach\bbl@calendars{%
2645        \bbl@ifunset{bbl@ca@##1}{%
2646          \chardef\atcatcode=\catcode`\@
2647          \catcode`\@=11\relax
2648          \InputIfFileExists{babel-ca-##1.tex}{}{}%
2649          \catcode`\@=\atcatcode
2650          \let\atcatcode\relax}%
2651        {}}%
2652    \fi
2653    % == frenchspacing ==
2654    \ifcase\bbl@howloaded\in@true\else\in@false\fi
2655    \ifin@\else\bbl@xin@{typography/frenchspacing}{\bbl@key@list}\fi
2656    \ifin@
2657      \bbl@extras@wrap{\\\bbl@pre@fs}%
2658        {\bbl@pre@fs}%
2659        {\bbl@post@fs}%
2660    \fi
2661    % == Release saved transforms ==
2662    \bbl@release@transforms\relax % \relax closes the last item.
2663    % == main ==
2664    \ifx\bbl@KVP@main\@nil  % Restore only if not 'main'
2665      \let\languagename\bbl@savelangname
2666      \chardef\localeid\bbl@savelocaleid\relax
2667    \fi}
```

Depending on whether or not the language exists (based on \date<language>), we define two macros. Remember \bbl@startcommands opens a group.

```
2668  \def\bbl@provide@new#1{%
2669    \@namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
2670    \@namedef{extras#1}{}%
2671    \@namedef{noextras#1}{}%
2672    \bbl@startcommands*{#1}{captions}%
2673      \ifx\bbl@KVP@captions\@nil %        and also if import, implicit
2674        \def\bbl@tempb##1{%             elt for \bbl@captionslist
2675          \ifx##1\@empty\else
2676            \bbl@exp{%
2677              \\\SetString\\##1{%
2678                \\\bbl@nocaption{\bbl@stripslash##1}{#1\bbl@stripslash##1}}}%
2679            \expandafter\bbl@tempb
2680          \fi}%
2681        \expandafter\bbl@tempb\bbl@captionslist\@empty
2682      \else
2683        \ifx\bbl@initoload\relax
2684          \bbl@read@ini{\bbl@KVP@captions}2%  % Here letters cat = 11
2685        \else
2686          \bbl@read@ini{\bbl@initoload}2%      % Same
2687        \fi
```

```
2688        \fi
2689    \StartBabelCommands*{#1}{date}%
2690      \ifx\bbl@KVP@import\@nil
2691        \bbl@exp{%
2692          \\\SetString\\\today{\\\bbl@nocaption{today}{#1today}}}%
2693      \else
2694        \bbl@savetoday
2695        \bbl@savedate
2696      \fi
2697    \bbl@endcommands
2698    \bbl@load@basic{#1}%
2699    % == hyphenmins == (only if new)
2700    \bbl@exp{%
2701      \gdef\<#1hyphenmins>{%
2702        {\bbl@ifunset{bbl@lfthm@#1}{2}{\bbl@cs{lfthm@#1}}}%
2703        {\bbl@ifunset{bbl@rgthm@#1}{3}{\bbl@cs{rgthm@#1}}}}}%
2704    % == hyphenrules (also in renew) ==
2705    \bbl@provide@hyphens{#1}%
2706    \ifx\bbl@KVP@main\@nil\else
2707      \expandafter\main@language\expandafter{#1}%
2708    \fi}
2709 %
2710 \def\bbl@provide@renew#1{%
2711    \ifx\bbl@KVP@captions\@nil\else
2712      \StartBabelCommands*{#1}{captions}%
2713        \bbl@read@ini{\bbl@KVP@captions}2%    % Here all letters cat = 11
2714      \EndBabelCommands
2715    \fi
2716    \ifx\bbl@KVP@import\@nil\else
2717      \StartBabelCommands*{#1}{date}%
2718        \bbl@savetoday
2719        \bbl@savedate
2720      \EndBabelCommands
2721    \fi
2722    % == hyphenrules (also in new) ==
2723    \ifx\bbl@lbkflag\@empty
2724      \bbl@provide@hyphens{#1}%
2725    \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.)

```
2726 \def\bbl@load@basic#1{%
2727    \ifcase\bbl@howloaded\or\or
2728      \ifcase\csname bbl@llevel@\languagename\endcsname
2729        \bbl@csarg\let{lname@\languagename}\relax
2730      \fi
2731    \fi
2732    \bbl@ifunset{bbl@lname@#1}%
2733      {\def\BabelBeforeIni##1##2{%
2734         \begingroup
2735           \let\bbl@ini@captions@aux\@gobbletwo
2736           \def\bbl@inidate ####1.####2.####3.####4\relax ####5####6{}%
2737           \bbl@read@ini{##1}1%
2738           \ifx\bbl@initoload\relax\endinput\fi
2739         \endgroup}%
2740       \begingroup        % boxed, to avoid extra spaces:
2741         \ifx\bbl@initoload\relax
2742           \bbl@input@texini{#1}%
2743         \else
2744           \setbox\z@\hbox{\BabelBeforeIni{\bbl@initoload}{}}%
2745         \fi
2746       \endgroup}%
```

```
2747      {}}
```

The hyphenrules option is handled with an auxiliary macro.

```
2748 \def\bbl@provide@hyphens#1{%
2749   \let\bbl@tempa\relax
2750   \ifx\bbl@KVP@hyphenrules\@nil\else
2751     \bbl@replace\bbl@KVP@hyphenrules{ }{,}%
2752     \bbl@foreach\bbl@KVP@hyphenrules{%
2753       \ifx\bbl@tempa\relax    % if not yet found
2754         \bbl@ifsamestring{##1}{+}%
2755           {{\bbl@exp{\\\addlanguage\<l@##1>}}}%
2756           {}%
2757         \bbl@ifunset{l@##1}%
2758           {}%
2759           {\bbl@exp{\let\bbl@tempa\<l@##1>}}%
2760       \fi}%
2761   \fi
2762   \ifx\bbl@tempa\relax %          if no opt or no language in opt found
2763     \ifx\bbl@KVP@import\@nil
2764       \ifx\bbl@initoload\relax\else
2765         \bbl@exp{%              and hyphenrules is not empty
2766           \\\bbl@ifblank{\bbl@cs{hyphr@#1}}%
2767             {}%
2768             {\let\\\bbl@tempa\<l@\bbl@cl{hyphr}>}}%
2769       \fi
2770     \else % if importing
2771       \bbl@exp{%                 and hyphenrules is not empty
2772         \\\bbl@ifblank{\bbl@cs{hyphr@#1}}%
2773           {}%
2774           {\let\\\bbl@tempa\<l@\bbl@cl{hyphr}>}}%
2775     \fi
2776   \fi
2777   \bbl@ifunset{bbl@tempa}%       ie, relax or undefined
2778     {\bbl@ifunset{l@#1}%         no hyphenrules found - fallback
2779       {\bbl@exp{\\\adddialect\<l@#1>\language}}%
2780       {}}%                       so, l@<lang> is ok - nothing to do
2781     {\bbl@exp{\\\adddialect\<l@#1>\bbl@tempa}}}% found in opt list or ini
```

The reader of `babel-...tex` files. We reset temporarily some catcodes.

```
2782 \def\bbl@input@texini#1{%
2783   \bbl@bsphack
2784     \bbl@exp{%
2785       \catcode`\\\%=14 \catcode`\\\\=0
2786       \catcode`\\\{=1  \catcode`\\\}=2
2787       \lowercase{\\\InputIfFileExists{babel-#1.tex}{}{}}%
2788       \catcode`\\\%=\the\catcode`\%\relax
2789       \catcode`\\\\=\the\catcode`\\\relax
2790       \catcode`\\\{=\the\catcode`\{\relax
2791       \catcode`\\\}=\the\catcode`\}\relax}%
2792   \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`.

```
2793 \def\bbl@iniline#1\bbl@iniline{%
2794   \@ifnextchar[\bbl@inisect{\@ifnextchar;\bbl@iniskip\bbl@inistore}#1\@@}% ]
2795 \def\bbl@inisect[#1]#2\@@{\def\bbl@section{#1}}
2796 \def\bbl@iniskip#1\@@{}%      if starts with ;
2797 \def\bbl@inistore#1=#2\@@{%      full (default)
2798   \bbl@trim@def\bbl@tempa{#1}%
2799   \bbl@trim\toks@{#2}%
2800   \bbl@xin@{;\bbl@section/\bbl@tempa;}{\bbl@key@list}%
2801   \ifin@\else
2802     \bbl@exp{%
```

```
2803        \\\g@addto@macro\\\bbl@inidata{%
2804          \\\bbl@elt{\bbl@section}{\bbl@tempa}{\the\toks@}}}%
2805    \fi}
2806 \def\bbl@inistore@min#1=#2\@@{%  minimal (maybe set in \bbl@read@ini)
2807    \bbl@trim@def\bbl@tempa{#1}%
2808    \bbl@trim\toks@{#2}%
2809    \bbl@xin@{.identification.}{.\bbl@section.}%
2810    \ifin@
2811      \bbl@exp{\\\g@addto@macro\\\bbl@inidata{%
2812        \\\bbl@elt{identification}{\bbl@tempa}{\the\toks@}}}%
2813    \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.

```
2814 \ifx\bbl@readstream\@undefined
2815    \csname newread\endcsname\bbl@readstream
2816 \fi
2817 \def\bbl@read@ini#1#2{%
2818    \global\let\bbl@extend@ini\@gobble
2819    \openin\bbl@readstream=babel-#1.ini
2820    \ifeof\bbl@readstream
2821      \bbl@error
2822        {There is no ini file for the requested language\\%
2823         (#1: \languagename). Perhaps you misspelled it or your\\%
2824         installation is not complete.}%
2825        {Fix the name or reinstall babel.}%
2826    \else
2827      % == Store ini data in \bbl@inidata ==
2828      \catcode`\[=12 \catcode`\]=12 \catcode`\==12 \catcode`\&=12
2829      \catcode`\;=12 \catcode`\|=12 \catcode`\%=14 \catcode`\-=12
2830      \bbl@info{Importing
2831                  \ifcase#2font and identification \or basic \fi
2832                   data for \languagename\\%
2833                from babel-#1.ini. Reported}%
2834      \ifnum#2=\z@
2835        \global\let\bbl@inidata\@empty
2836        \let\bbl@inistore\bbl@inistore@min    % Remember it's local
2837      \fi
2838      \def\bbl@section{identification}%
2839      \bbl@exp{\\\bbl@inistore tag.ini=#1\\\@@}%
2840      \bbl@inistore load.level=#2\@@
2841      \loop
2842      \if T\ifeof\bbl@readstream F\fi T\relax % Trick, because inside \loop
2843        \endlinechar\m@ne
2844        \read\bbl@readstream to \bbl@line
2845        \endlinechar`\^^M
2846        \ifx\bbl@line\@empty\else
2847          \expandafter\bbl@iniline\bbl@line\bbl@iniline
2848        \fi
2849      \repeat
2850      % == Process stored data ==
2851      \bbl@csarg\xdef{lini@\languagename}{#1}%
2852      \bbl@read@ini@aux
2853      % == 'Export' data ==
2854      \bbl@ini@exports{#2}%
2855      \global\bbl@csarg\let{inidata@\languagename}\bbl@inidata
2856      \global\let\bbl@inidata\@empty
2857      \bbl@exp{\\\bbl@add@list\\\bbl@ini@loaded{\languagename}}%
2858      \bbl@toglobal\bbl@ini@loaded
```

```
2859   \fi}
2860 \def\bbl@read@ini@aux{%
2861   \let\bbl@savestrings\@empty
2862   \let\bbl@savetoday\@empty
2863   \let\bbl@savedate\@empty
2864   \def\bbl@elt##1##2##3{%
2865     \def\bbl@section{##1}%
2866     \in@{=date.}{=##1}% Find a better place
2867     \ifin@
2868       \bbl@ifunset{bbl@inikv@##1}%
2869         {\bbl@ini@calendar{##1}}%
2870         {}%
2871     \fi
2872     \in@{=identification/extension.}{=##1/##2}%
2873     \ifin@
2874       \bbl@ini@extension{##2}%
2875     \fi
2876     \bbl@ifunset{bbl@inikv@##1}{}%
2877       {\csname bbl@inikv@##1\endcsname{##2}{##3}}}%
2878   \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.

```
2879 \def\bbl@extend@ini@aux#1{%
2880   \bbl@startcommands*{#1}{captions}%
2881     % Activate captions/... and modify exports
2882     \bbl@csarg\def{inikv@captions.licr}##1##2{%
2883       \setlocalecaption{#1}{##1}{##2}}%
2884     \def\bbl@inikv@captions##1##2{%
2885       \bbl@ini@captions@aux{##1}{##2}}%
2886     \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2887     \def\bbl@exportkey##1##2##3{%
2888       \bbl@ifunset{bbl@@kv@##2}{}%
2889         {\expandafter\ifx\csname bbl@@kv@##2\endcsname\@empty\else
2890           \bbl@exp{\global\let\<bbl@##1@\languagename>\<bbl@@kv@##2>}%
2891         \fi}}%
2892     % As with \bbl@read@ini, but with some changes
2893     \bbl@read@ini@aux
2894     \bbl@ini@exports\tw@
2895     % Update inidata@lang by pretending the ini is read.
2896     \def\bbl@elt##1##2##3{%
2897       \def\bbl@section{##1}%
2898       \bbl@iniline##2=##3\bbl@iniline}%
2899     \csname bbl@inidata@#1\endcsname
2900     \global\bbl@csarg\let{inidata@#1}\bbl@inidata
2901   \StartBabelCommands*{#1}{date}% And from the import stuff
2902     \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2903     \bbl@savetoday
2904     \bbl@savedate
2905   \bbl@endcommands}
```

A somewhat hackish tool to handle calendar sections. TODO. To be improved.

```
2906 \def\bbl@ini@calendar#1{%
2907 \lowercase{\def\bbl@tempa{=#1=}}%
2908 \bbl@replace\bbl@tempa{=date.gregorian}{}%
2909 \bbl@replace\bbl@tempa{=date.}{}%
2910 \in@{.licr=}{#1=}%
2911 \ifin@
2912   \ifcase\bbl@engine
2913     \bbl@replace\bbl@tempa{.licr=}{}%
2914   \else
2915     \let\bbl@tempa\relax
2916   \fi
2917 \fi
```

```
2918  \ifx\bbl@tempa\relax\else
2919    \bbl@replace\bbl@tempa{=}{}%
2920    \ifx\bbl@tempa\@empty\else
2921      \xdef\bbl@calendars{,\bbl@tempa}%
2922    \fi
2923    \bbl@exp{%
2924      \def\<bbl@inikv@#1>####1####2{%
2925        \\\bbl@inidate####1...\relax{####2}{\bbl@tempa}}}%
2926  \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).

```
2927  \def\bbl@renewinikey#1/#2\@@#3{%
2928    \edef\bbl@tempa{\zap@space #1 \@empty}%    section
2929    \edef\bbl@tempb{\zap@space #2 \@empty}%    key
2930    \bbl@trim\toks@{#3}%                       value
2931    \bbl@exp{%
2932      \edef\\\bbl@key@list{\bbl@key@list \bbl@tempa/\bbl@tempb;}%
2933      \\\g@addto@macro\\\bbl@inidata{%
2934        \\\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.

```
2935  \def\bbl@exportkey#1#2#3{%
2936    \bbl@ifunset{bbl@@kv@#2}%
2937      {\bbl@csarg\gdef{#1@\languagename}{#3}}%
2938      {\expandafter\ifx\csname bbl@@kv@#2\endcsname\@empty
2939        \bbl@csarg\gdef{#1@\languagename}{#3}%
2940      \else
2941        \bbl@exp{\global\let\<bbl@#1@\languagename>\<bbl@@kv@#2>}%
2942      \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.

```
2943  \def\bbl@iniwarning#1{%
2944    \bbl@ifunset{bbl@@kv@identification.warning#1}{}%
2945      {\bbl@warning{%
2946        From babel-\bbl@cs{lini@\languagename}.ini:\\%
2947        \bbl@cs{@kv@identification.warning#1}\\%
2948        Reported }}}
2949  %
2950  \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.

```
2951  \def\bbl@ini@extension#1{%
2952    \def\bbl@tempa{#1}%
2953    \bbl@replace\bbl@tempa{extension.}{}%
2954    \bbl@replace\bbl@tempa{.tag.bcp47}{}%
2955    \bbl@ifunset{bbl@info@#1}%
2956      {\bbl@csarg\xdef{info@#1}{ext/\bbl@tempa}%
2957      \bbl@exp{%
2958        \\\g@addto@macro\\\bbl@moreinfo{%
2959          \\\bbl@exportkey{ext/\bbl@tempa}{identification.#1}{}}}}%
2960      {}}
2961  \let\bbl@moreinfo\@empty
2962  %
2963  \def\bbl@ini@exports#1{%
2964    % Identification always exported
2965    \bbl@iniwarning{}%
2966    \ifcase\bbl@engine
```

```
2967      \bbl@iniwarning{.pdflatex}%
2968    \or
2969      \bbl@iniwarning{.lualatex}%
2970    \or
2971      \bbl@iniwarning{.xelatex}%
2972    \fi%
2973    \bbl@exportkey{llevel}{identification.load.level}{}%
2974    \bbl@exportkey{elname}{identification.name.english}{}%
2975    \bbl@exp{\\\bbl@exportkey{lname}{identification.name.opentype}%
2976      {\csname bbl@elname@\languagename\endcsname}}%
2977    \bbl@exportkey{tbcp}{identification.tag.bcp47}{}%
2978    \bbl@exportkey{lbcp}{identification.language.tag.bcp47}{}%
2979    \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
2980    \bbl@exportkey{esname}{identification.script.name}{}%
2981    \bbl@exp{\\\bbl@exportkey{sname}{identification.script.name.opentype}%
2982      {\csname bbl@esname@\languagename\endcsname}}%
2983    \bbl@exportkey{sbcp}{identification.script.tag.bcp47}{}%
2984    \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
2985    \bbl@exportkey{rbcp}{identification.region.tag.bcp47}{}%
2986    \bbl@exportkey{vbcp}{identification.variant.tag.bcp47}{}%
2987    \bbl@moreinfo
2988    % Also maps bcp47 -> languagename
2989    \ifbbl@bcptoname
2990      \bbl@csarg\xdef{bcp@map@\bbl@cl{tbcp}}{\languagename}%
2991    \fi
2992    % Conditional
2993    \ifnum#1>\z@          % 0 = only info, 1, 2 = basic, (re)new
2994      \bbl@exportkey{calpr}{date.calendar.preferred}{}%
2995      \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
2996      \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
2997      \bbl@exportkey{lfthm}{typography.lefthyphenmin}{2}%
2998      \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
2999      \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
3000      \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
3001      \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
3002      \bbl@exportkey{intsp}{typography.intraspace}{}%
3003      \bbl@exportkey{frspc}{typography.frenchspacing}{u}%
3004      \bbl@exportkey{chrng}{characters.ranges}{}%
3005      \bbl@exportkey{quote}{characters.delimiters.quotes}{}%
3006      \bbl@exportkey{dgnat}{numbers.digits.native}{}%
3007      \ifnum#1=\tw@            % only (re)new
3008        \bbl@exportkey{rqtex}{identification.require.babel}{}%
3009        \bbl@toglobal\bbl@savetoday
3010        \bbl@toglobal\bbl@savedate
3011        \bbl@savestrings
3012      \fi
3013    \fi}
```

A shared handler for key=val lines to be stored in \bbl@@kv@<section>.<key>.

```
3014 \def\bbl@inikv#1#2{%        key=value
3015   \toks@{#2}%               This hides #'s from ini values
3016   \bbl@csarg\edef{@kv@\bbl@section.#1}{\the\toks@}}
```

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

```
3017 \let\bbl@inikv@identification\bbl@inikv
3018 \let\bbl@inikv@date\bbl@inikv
3019 \let\bbl@inikv@typography\bbl@inikv
3020 \let\bbl@inikv@characters\bbl@inikv
3021 \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'.

```
3022 \def\bbl@inikv@counters#1#2{%
```

```
3023    \bbl@ifsamestring{#1}{digits}%
3024      {\bbl@error{The counter name 'digits' is reserved for mapping\\%
3025                 decimal digits}%
3026                {Use another name.}}%
3027      {}%
3028    \def\bbl@tempc{#1}%
3029    \bbl@trim@def{\bbl@tempb*}{#2}%
3030    \in@{.1$}{#1$}%
3031    \ifin@
3032      \bbl@replace\bbl@tempc{.1}{}%
3033      \bbl@csarg\protected@xdef{cntr@\bbl@tempc @\languagename}{%
3034        \noexpand\bbl@alphnumeral{\bbl@tempc}}%
3035    \fi
3036    \in@{.F.}{#1}%
3037    \ifin@\else\in@{.S.}{#1}\fi
3038    \ifin@
3039      \bbl@csarg\protected@xdef{cntr@#1@\languagename}{\bbl@tempb*}%
3040    \else
3041      \toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa
3042      \expandafter\bbl@buildifcase\bbl@tempb* \\ % Space after \\
3043      \bbl@csarg{\global\expandafter\let}{cntr@#1@\languagename}\bbl@tempa
3044    \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.

```
3045  \ifcase\bbl@engine
3046    \bbl@csarg\def{inikv@captions.licr}#1#2{%
3047      \bbl@ini@captions@aux{#1}{#2}}
3048  \else
3049    \def\bbl@inikv@captions#1#2{%
3050      \bbl@ini@captions@aux{#1}{#2}}
3051  \fi
```

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

```
3052  \def\bbl@ini@captions@template#1#2{% string language tempa=capt-name
3053    \bbl@replace\bbl@tempa{.template}{}%
3054    \def\bbl@toreplace{#1{}}%
3055    \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
3056    \bbl@replace\bbl@toreplace{[[}{\csname}%
3057    \bbl@replace\bbl@toreplace{[}{\csname the}%
3058    \bbl@replace\bbl@toreplace{]]}{name\endcsname{}}%
3059    \bbl@replace\bbl@toreplace{]}{\endcsname{}}%
3060    \bbl@xin@{,\bbl@tempa,}{,chapter,appendix,part,}%
3061    \ifin@
3062      \@nameuse{bbl@patch\bbl@tempa}%
3063      \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
3064    \fi
3065    \bbl@xin@{,\bbl@tempa,}{,figure,table,}%
3066    \ifin@
3067      \toks@\expandafter{\bbl@toreplace}%
3068      \bbl@exp{\gdef\<fnum@\bbl@tempa>{\the\toks@}}%
3069    \fi}
3070  \def\bbl@ini@captions@aux#1#2{%
3071    \bbl@trim@def\bbl@tempa{#1}%
3072    \bbl@xin@{.template}{\bbl@tempa}%
3073    \ifin@
3074      \bbl@ini@captions@template{#2}\languagename
3075    \else
3076      \bbl@ifblank{#2}%
3077        {\bbl@exp{%
3078          \toks@{\\\bbl@nocaption{\bbl@tempa}{\languagename\bbl@tempa name}}}}%
3079        {\bbl@trim\toks@{#2}}%
3080      \bbl@exp{%
```

125

```
3081      \\\bbl@add\\\bbl@savestrings{%
3082        \\\SetString\<\bbl@tempa name>{\the\toks@}}}%
3083    \toks@\expandafter{\bbl@captionslist}%
3084    \bbl@exp{\\\in@{<\bbl@tempa name>}{\the\toks@}}%
3085    \ifin@\else
3086      \bbl@exp{%
3087        \\\bbl@add\<bbl@extracaps@\languagename>{\<\bbl@tempa name>}%
3088        \\\bbl@toglobal\<bbl@extracaps@\languagename>}%
3089    \fi
3090  \fi}
```

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

```
3091 \def\bbl@list@the{%
3092    part,chapter,section,subsection,subsubsection,paragraph,%
3093    subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,%
3094    table,page,footnote,mpfootnote,mpfn}
3095 \def\bbl@map@cnt#1{%  #1:roman,etc, // #2:enumi,etc
3096    \bbl@ifunset{bbl@map@#1@\languagename}%
3097      {\@nameuse{#1}}%
3098      {\@nameuse{bbl@map@#1@\languagename}}}
3099 \def\bbl@inikv@labels#1#2{%
3100    \in@{.map}{#1}%
3101    \ifin@
3102      \ifx\bbl@KVP@labels\@nil\else
3103        \bbl@xin@{ map }{ \bbl@KVP@labels\space}%
3104        \ifin@
3105          \def\bbl@tempc{#1}%
3106          \bbl@replace\bbl@tempc{.map}{}%
3107          \in@{,#2,}{,arabic,roman,Roman,alph,Alph,fnsymbol,}%
3108          \bbl@exp{%
3109            \gdef\<bbl@map@\bbl@tempc @\languagename>%
3110              {\ifin@\<#2>\else\\\localecounter{#2}\fi}}%
3111          \bbl@foreach\bbl@list@the{%
3112            \bbl@ifunset{the##1}{}%
3113              {\bbl@exp{\let\\\bbl@tempd\<the##1>}%
3114               \bbl@exp{%
3115                 \\\bbl@sreplace\<the##1>%
3116                   {\<\bbl@tempc>{##1}}{\\\bbl@map@cnt{\bbl@tempc}{##1}}%
3117                 \\\bbl@sreplace\<the##1>%
3118                   {\<\@empty @\bbl@tempc>\<c@##1>}{\\\bbl@map@cnt{\bbl@tempc}{##1}}}%
3119               \expandafter\ifx\csname the##1\endcsname\bbl@tempd\else
3120                 \toks@\expandafter\expandafter\expandafter{%
3121                   \csname the##1\endcsname}%
3122                 \expandafter\xdef\csname the##1\endcsname{{\the\toks@}}%
3123              \fi}}%
3124        \fi
3125    \fi
3126  %
3127  \else
3128    %
3129    % The following code is still under study. You can test it and make
3130    % suggestions. Eg, enumerate.2 = ([enumi]).([enumii]). It's
3131    % language dependent.
3132    \in@{enumerate.}{#1}%
3133    \ifin@
3134      \def\bbl@tempa{#1}%
3135      \bbl@replace\bbl@tempa{enumerate.}{}%
3136      \def\bbl@toreplace{#2}%
3137      \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
3138      \bbl@replace\bbl@toreplace{[}{\csname the}%
3139      \bbl@replace\bbl@toreplace{]}{\endcsname{}}%
3140      \toks@\expandafter{\bbl@toreplace}%
3141      % TODO. Execute only once:
```

```
3142        \bbl@exp{%
3143          \\\bbl@add\<extras\languagename>{%
3144            \\\babel@save\<labelenum\romannumeral\bbl@tempa>%
3145            \def\<labelenum\romannumeral\bbl@tempa>{\the\toks@}}%
3146          \\\bbl@toglobal\<extras\languagename>}%
3147      \fi
3148    \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.

```
3149 \def\bbl@chaptype{chapter}
3150 \ifx\@makechapterhead\@undefined
3151   \let\bbl@patchchapter\relax
3152 \else\ifx\thechapter\@undefined
3153   \let\bbl@patchchapter\relax
3154 \else\ifx\ps@headings\@undefined
3155   \let\bbl@patchchapter\relax
3156 \else
3157   \def\bbl@patchchapter{%
3158     \global\let\bbl@patchchapter\relax
3159     \gdef\bbl@chfmt{%
3160       \bbl@ifunset{bbl@\bbl@chaptype fmt@\languagename}%
3161         {\@chapapp\space\thechapter}
3162         {\@nameuse{bbl@\bbl@chaptype fmt@\languagename}}}
3163     \bbl@add\appendix{\def\bbl@chaptype{appendix}}% Not harmful, I hope
3164     \bbl@sreplace\ps@headings{\@chapapp\ \thechapter}{\bbl@chfmt}%
3165     \bbl@sreplace\chaptermark{\@chapapp\ \thechapter}{\bbl@chfmt}%
3166     \bbl@sreplace\@makechapterhead{\@chapapp\space\thechapter}{\bbl@chfmt}%
3167     \bbl@toglobal\appendix
3168     \bbl@toglobal\ps@headings
3169     \bbl@toglobal\chaptermark
3170     \bbl@toglobal\@makechapterhead}
3171   \let\bbl@patchappendix\bbl@patchchapter
3172 \fi\fi\fi
3173 \ifx\@part\@undefined
3174   \let\bbl@patchpart\relax
3175 \else
3176   \def\bbl@patchpart{%
3177     \global\let\bbl@patchpart\relax
3178     \gdef\bbl@partformat{%
3179       \bbl@ifunset{bbl@partfmt@\languagename}%
3180         {\partname\nobreakspace\thepart}
3181         {\@nameuse{bbl@partfmt@\languagename}}}
3182     \bbl@sreplace\@part{\partname\nobreakspace\thepart}{\bbl@partformat}%
3183     \bbl@toglobal\@part}
3184 \fi
```

**Date.** TODO. Document

```
3185 % Arguments are _not_ protected.
3186 \let\bbl@calendar\@empty
3187 \DeclareRobustCommand\localedate[1][]{\bbl@localedate{#1}}
3188 \def\bbl@localedate#1#2#3#4{%
3189   \begingroup
3190     \edef\bbl@tempe{#1}%
3191     \edef\bbl@they{#2}%
3192     \edef\bbl@them{#3}%
3193     \edef\bbl@thed{#4}%
3194     \ifx\bbl@tempe\@empty
3195       \bbl@ifunset{bbl@calpr@\languagename}{}%
3196         {\edef\bbl@tempe{convert,calendar=\bbl@cl{calpr}}%
3197          \@nameuse{regex\string_replace\string_once\string:nnN}%
3198            {\ .*}{}\bbl@tempe}% TODO. Must be more efficient
```

```
3199    \fi
3200    \bbl@replace\bbl@tempe{ }{}%
3201    \bbl@replace\bbl@tempe{convert}{convert=}%
3202    \let\bbl@ld@calendar\@empty
3203    \let\bbl@ld@variant\@empty
3204    \let\bbl@ld@convert\relax
3205    \def\bbl@tempb##1=##2\@@{\@namedef{bbl@ld@##1}{##2}}%
3206    \bbl@foreach\bbl@tempe{\bbl@tempb##1\@@}%
3207    \ifx\bbl@ld@convert\relax\else
3208      \let\bbl@ld@convert\bbl@ld@calendar
3209      \bbl@replace\bbl@ld@convert{gregorian}{}%
3210      \ifx\bbl@ld@convert\@empty\else
3211        \let\bbl@ld@convert\bbl@ld@calendar % For conversion
3212        \@nameuse{regex\string_replace\string_once\string:nnN}%
3213          {\..*}{}\bbl@ld@convert
3214        \ifx\bbl@ld@convert\@empty\else
3215          \babelcalendar[\bbl@they-\bbl@them-\bbl@thed]%
3216            {\bbl@ld@convert}\bbl@they\bbl@them\bbl@thed
3217        \fi
3218      \fi
3219    \fi
3220    \bbl@replace\bbl@ld@calendar{gregorian}{}%
3221    \@nameuse{bbl@precalendar}%
3222    \edef\bbl@calendar{% Used in \month..., too
3223      \bbl@ld@calendar
3224      \ifx\bbl@ld@variant\@empty\else
3225        .\bbl@ld@variant
3226      \fi}%
3227    \bbl@cased
3228      {\@nameuse{bbl@date@\languagename @\bbl@calendar}%
3229        \bbl@they\bbl@them\bbl@thed}%
3230  \endgroup}
3231 % eg: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
3232 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
3233    \bbl@trim@def\bbl@tempa{#1.#2}%
3234    \bbl@ifsamestring{\bbl@tempa}{months.wide}%        to savedate
3235      {\bbl@trim@def\bbl@tempa{#3}%
3236      \bbl@trim\toks@{#5}%
3237      \@temptokena\expandafter{\bbl@savedate}%
3238      \bbl@exp{%   Reverse order - in ini last wins
3239        \def\\\bbl@savedate{%
3240          \\\SetString\<month\romannumeral\bbl@tempa#6name>{\the\toks@}%
3241          \the\@temptokena}}%
3242    {\bbl@ifsamestring{\bbl@tempa}{date.long}%      defined now
3243      {\lowercase{\def\bbl@tempb{#6}}%
3244      \bbl@trim@def\bbl@toreplace{#5}%
3245      \bbl@TG@@date
3246      \global\bbl@csarg\let{date@\languagename @\bbl@tempb}\bbl@toreplace
3247      \ifx\bbl@savetoday\@empty
3248        \bbl@exp{% TODO. Move to a better place.
3249          \gdef\<\languagename date>{\\\protect\<\languagename date >}%
3250          \gdef\<\languagename date >####1####2####3{%
3251            \\\bbl@usedategrouptrue
3252            \<bbl@ensure@\languagename>{%
3253              \\\localedate{####1}{####2}{####3}}}%
3254          \def\\\bbl@savetoday{%
3255            \\\SetString\\\today{%
3256              \<\languagename date>%
3257                {\\\the\year}{\\\the\month}{\\\the\day}}}}%
3258      \fi}%
3259      {}}}
```

**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).

```
3260 \let\bbl@calendar\@empty
3261 \newcommand\babelcalendar[2][\the\year-\the\month-\the\day]{%
3262   \@nameuse{bbl@ca@#2}#1\@@}
3263 \newcommand\BabelDateSpace{\nobreakspace}
3264 \newcommand\BabelDateDot{.\@}  % TODO. \let instead of repeating
3265 \newcommand\BabelDated[1]{{\number#1}}
3266 \newcommand\BabelDatedd[1]{{\ifnum#1<10 0\fi\number#1}}
3267 \newcommand\BabelDateM[1]{{\number#1}}
3268 \newcommand\BabelDateMM[1]{{\ifnum#1<10 0\fi\number#1}}
3269 \newcommand\BabelDateMMMM[1]{{%
3270   \csname month\romannumeral#1\bbl@calendar name\endcsname}}
3271 \newcommand\BabelDatey[1]{{\number#1}}%
3272 \newcommand\BabelDateyy[1]{{%
3273   \ifnum#1<10 0\number#1 %
3274   \else\ifnum#1<100 \number#1 %
3275   \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3276   \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
3277   \else
3278     \bbl@error
3279       {Currently two-digit years are restricted to the\\
3280        range 0-9999.}%
3281       {There is little you can do. Sorry.}%
3282   \fi\fi\fi\fi}}
3283 \newcommand\BabelDateyyyy[1]{{\number#1}} % TODO - add leading 0
3284 \def\bbl@replace@finish@iii#1{%
3285   \bbl@exp{\def\\#1####1####2####3{\the\toks@}}}
3286 \def\bbl@TG@@date{%
3287   \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace{}}%
3288   \bbl@replace\bbl@toreplace{[.]}{\BabelDateDot{}}%
3289   \bbl@replace\bbl@toreplace{[d]}{\BabelDated{####3}}%
3290   \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{####3}}%
3291   \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{####2}}%
3292   \bbl@replace\bbl@toreplace{[MM]}{\BabelDateMM{####2}}%
3293   \bbl@replace\bbl@toreplace{[MMMM]}{\BabelDateMMMM{####2}}%
3294   \bbl@replace\bbl@toreplace{[y]}{\BabelDatey{####1}}%
3295   \bbl@replace\bbl@toreplace{[yy]}{\BabelDateyy{####1}}%
3296   \bbl@replace\bbl@toreplace{[yyyy]}{\BabelDateyyyy{####1}}%
3297   \bbl@replace\bbl@toreplace{[y|}{\bbl@datecntr[####1|}%
3298   \bbl@replace\bbl@toreplace{[m|}{\bbl@datecntr[####2|}%
3299   \bbl@replace\bbl@toreplace{[d|}{\bbl@datecntr[####3|}%
3300   \bbl@replace@finish@iii\bbl@toreplace}
3301 \def\bbl@datecntr{\expandafter\bbl@xdatecntr\expandafter}
3302 \def\bbl@xdatecntr[#1|#2]{\localenumeral{#2}{#1}}
```

**Transforms.**

```
3303 \let\bbl@release@transforms\@empty
3304 \@namedef{bbl@inikv@transforms.prehyphenation}{%
3305   \bbl@transforms\babelprehyphenation}
3306 \@namedef{bbl@inikv@transforms.posthyphenation}{%
3307   \bbl@transforms\babelposthyphenation}
3308 \def\bbl@transforms@aux#1#2#3#4,#5\relax{%
3309   #1[#2]{#3}{#4}{#5}}
3310 \begingroup %  A hack. TODO. Don't require an specific order
3311   \catcode`\%=12
3312   \catcode`\&=14
3313   \gdef\bbl@transforms#1#2#3{&%
3314     \ifx\bbl@KVP@transforms\@nil\else
3315       \directlua{
3316         local str = [==[#2]==]
3317         str = str:gsub('%.%d+%.%d+$', '')
```

129

```
3318          tex.print([[\def\string\babeltempa{]] .. str .. [[}]])
3319        }&%
3320      \bbl@xin@{,\babeltempa,}{,\bbl@KVP@transforms,}&%
3321      \ifin@
3322        \in@{.0$}{#2$}&%
3323        \ifin@
3324          \directlua{
3325            local str = string.match([[\bbl@KVP@transforms]],
3326                        '%(([^%(]-)%)[^%]]-\babeltempa')
3327            if str == nil then
3328              tex.print([[\def\string\babeltempb{}]])
3329            else
3330              tex.print([[\def\string\babeltempb{,attribute=]] .. str .. [[}]])
3331            end
3332          }
3333          \toks@{#3}&%
3334          \bbl@exp{&%
3335            \\\g@addto@macro\\\bbl@release@transforms{&%
3336              \relax  &% Closes previous \bbl@transforms@aux
3337              \\\bbl@transforms@aux
3338                \\#1{label=\babeltempa\babeltempb}{\languagename}{\the\toks@}}}&%
3339        \else
3340          \g@addto@macro\bbl@release@transforms{, {#3}}&%
3341        \fi
3342      \fi
3343    \fi}
3344 \endgroup
```

Language and Script values to be used when defining a font or setting the direction are set with the following macros.

```
3345 \def\bbl@provide@lsys#1{%
3346   \bbl@ifunset{bbl@lname@#1}%
3347     {\bbl@load@info{#1}}%
3348     {}%
3349   \bbl@csarg\let{lsys@#1}\@empty
3350   \bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}{}%
3351   \bbl@ifunset{bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{}%
3352   \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
3353   \bbl@ifunset{bbl@lname@#1}{}%
3354     {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
3355   \ifcase\bbl@engine\or\or
3356     \bbl@ifunset{bbl@prehc@#1}{}%
3357       {\bbl@exp{\\\bbl@ifblank{\bbl@cs{prehc@#1}}}%
3358         {}%
3359         {\ifx\bbl@xenohyph\@undefined
3360            \let\bbl@xenohyph\bbl@xenohyph@d
3361            \ifx\AtBeginDocument\@notprerr
3362              \expandafter\@secondoftwo  % to execute right now
3363            \fi
3364            \AtBeginDocument{%
3365              \bbl@patchfont{\bbl@xenohyph}%
3366              \expandafter\selectlanguage\expandafter{\languagename}}%
3367         \fi}}%
3368   \fi
3369   \bbl@csarg\bbl@toglobal{lsys@#1}}
3370 \def\bbl@xenohyph@d{%
3371   \bbl@ifset{bbl@prehc@\languagename}%
3372     {\ifnum\hyphenchar\font=\defaulthyphenchar
3373        \iffontchar\font\bbl@cl{prehc}\relax
3374          \hyphenchar\font\bbl@cl{prehc}\relax
3375        \else\iffontchar\font"200B
3376          \hyphenchar\font"200B
3377        \else
```

```
3378        \bbl@warning
3379          {Neither 0 nor ZERO WIDTH SPACE are available\\%
3380           in the current font, and therefore the hyphen\\%
3381           will be printed. Try changing the fontspec's\\%
3382           'HyphenChar' to another value, but be aware\\%
3383           this setting is not safe (see the manual)}%
3384         \hyphenchar\font\defaulthyphenchar
3385       \fi\fi
3386     \fi}%
3387   {\hyphenchar\font\defaulthyphenchar}}
3388 % \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).

```
3389 \def\bbl@load@info#1{%
3390   \def\BabelBeforeIni##1##2{%
3391     \begingroup
3392       \bbl@read@ini{##1}0%
3393       \endinput           % babel- .tex may contain onlypreamble's
3394     \endgroup}%              boxed, to avoid extra spaces:
3395   {\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.

```
3396 \def\bbl@setdigits#1#2#3#4#5{%
3397   \bbl@exp{%
3398     \def\<\languagename digits>####1{%        ie, \langdigits
3399       \<bbl@digits@\languagename>####1\\\@nil}%
3400     \let\<bbl@cntr@digits@\languagename>\<\languagename digits>%
3401     \def\<\languagename counter>####1{%       ie, \langcounter
3402       \\\expandafter\<bbl@counter@\languagename>%
3403       \\\csname c@####1\endcsname}%
3404     \def\<bbl@counter@\languagename>####1{% ie, \bbl@counter@lang
3405       \\\expandafter\<bbl@digits@\languagename>%
3406       \\\number####1\\\@nil}}%
3407   \def\bbl@tempa##1##2##3##4##5{%
3408     \bbl@exp{%    Wow, quite a lot of hashes! :-(
3409       \def\<bbl@digits@\languagename>########1{%
3410         \\\ifx########1\\\@nil               % ie, \bbl@digits@lang
3411         \\\else
3412           \\\ifx0########1#1%
3413           \\\else\\\ifx1########1#2%
3414           \\\else\\\ifx2########1#3%
3415           \\\else\\\ifx3########1#4%
3416           \\\else\\\ifx4########1#5%
3417           \\\else\\\ifx5########1##1%
3418           \\\else\\\ifx6########1##2%
3419           \\\else\\\ifx7########1##3%
3420           \\\else\\\ifx8########1##4%
3421           \\\else\\\ifx9########1##5%
3422           \\\else########1%
3423           \\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi
3424           \\\expandafter\<bbl@digits@\languagename>%
3425         \\\fi}}}%
3426   \bbl@tempa}
```

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

```
3427 \def\bbl@buildifcase#1 {% Returns \bbl@tempa, requires \toks@={}
3428   \ifx\\#1%               % \\ before, in case #1 is multiletter
3429     \bbl@exp{%
3430       \def\\\bbl@tempa####1{%
```

```
3431          \<ifcase>####1\space\the\toks@\<else>\\\@ctrerr\<fi>}}%
3432    \else
3433      \toks@\expandafter{\the\toks@\or #1}%
3434      \expandafter\bbl@buildifcase
3435    \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).

```
3436 \newcommand\localenumeral[2]{\bbl@cs{cntr@#1@\languagename}{#2}}
3437 \def\bbl@localecntr#1#2{\localenumeral{#2}{#1}}
3438 \newcommand\localecounter[2]{%
3439    \expandafter\bbl@localecntr
3440    \expandafter{\number\csname c@#2\endcsname}{#1}}
3441 \def\bbl@alphnumeral#1#2{%
3442    \expandafter\bbl@alphnumeral@i\number#2 76543210\@@{#1}}
3443 \def\bbl@alphnumeral@i#1#2#3#4#5#6#7#8\@@#9{%
3444    \ifcase\@car#8\@nil\or    % Currenty <10000, but prepared for bigger
3445      \bbl@alphnumeral@ii{#9}000000#1\or
3446      \bbl@alphnumeral@ii{#9}00000#1#2\or
3447      \bbl@alphnumeral@ii{#9}0000#1#2#3\or
3448      \bbl@alphnumeral@ii{#9}000#1#2#3#4\else
3449      \bbl@alphnum@invalid{>9999}%
3450    \fi}
3451 \def\bbl@alphnumeral@ii#1#2#3#4#5#6#7#8{%
3452    \bbl@ifunset{bbl@cntr@#1.F.\number#5#6#7#8@\languagename}%
3453      {\bbl@cs{cntr@#1.4@\languagename}#5%
3454        \bbl@cs{cntr@#1.3@\languagename}#6%
3455        \bbl@cs{cntr@#1.2@\languagename}#7%
3456        \bbl@cs{cntr@#1.1@\languagename}#8%
3457      \ifnum#6#7#8>\z@ % TODO. An ad hoc rule for Greek. Ugly.
3458        \bbl@ifunset{bbl@cntr@#1.S.321@\languagename}{}%
3459          {\bbl@cs{cntr@#1.S.321@\languagename}}%
3460      \fi}%
3461      {\bbl@cs{cntr@#1.F.\number#5#6#7#8@\languagename}}}
3462 \def\bbl@alphnum@invalid#1{%
3463    \bbl@error{Alphabetic numeral too large (#1)}%
3464      {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.

```
3465 \def\bbl@localeinfo#1#2{%
3466    \bbl@ifunset{bbl@info@#2}{#1}%
3467      {\bbl@ifunset{bbl@\csname bbl@info@#2\endcsname @\languagename}{#1}%
3468        {\bbl@cs{\csname bbl@info@#2\endcsname @\languagename}}}}
3469 \newcommand\localeinfo[1]{%
3470    \ifx*#1\@empty    % TODO. A bit hackish to make it expandable.
3471      \bbl@afterelse\bbl@localeinfo{}%
3472    \else
3473      \bbl@localeinfo
3474        {\bbl@error{I've found no info for the current locale.\\%
3475                    The corresponding ini file has not been loaded\\%
3476                    Perhaps it doesn't exist}%
3477                   {See the manual for details.}}%
3478        {#1}%
3479    \fi}
3480 % \@namedef{bbl@info@name.locale}{lcname}
3481 \@namedef{bbl@info@tag.ini}{lini}
3482 \@namedef{bbl@info@name.english}{elname}
3483 \@namedef{bbl@info@name.opentype}{lname}
3484 \@namedef{bbl@info@tag.bcp47}{tbcp}
3485 \@namedef{bbl@info@language.tag.bcp47}{lbcp}
```

```
3486 \@namedef{bbl@info@tag.opentype}{lotf}
3487 \@namedef{bbl@info@script.name}{esname}
3488 \@namedef{bbl@info@script.name.opentype}{sname}
3489 \@namedef{bbl@info@script.tag.bcp47}{sbcp}
3490 \@namedef{bbl@info@script.tag.opentype}{sotf}
3491 \@namedef{bbl@info@region.tag.bcp47}{rbcp}
3492 \@namedef{bbl@info@variant.tag.bcp47}{vbcp}
3493 % Extensions are dealt with in a special way
3494 % Now, an internal \LaTeX{} macro:
3495 \providecommand\BCPdata[1]{\localeinfo*{#1.tag.bcp47}}
```

With version 3.75 \BabelEnsureInfo is executed always, but there is an option to disable it.

```
3496 ⟨*More package options⟩ ≡
3497 \DeclareOption{ensureinfo=off}{}
3498 ⟨/More package options⟩
3499 %
3500 \let\bbl@ensureinfo\@gobble
3501 \newcommand\BabelEnsureInfo{%
3502   \ifx\InputIfFileExists\@undefined\else
3503     \def\bbl@ensureinfo##1{%
3504       \bbl@ifunset{bbl@lname@##1}{\bbl@load@info{##1}}{}}%
3505   \fi
3506   \bbl@foreach\bbl@loaded{{%
3507     \def\languagename{##1}%
3508     \bbl@ensureinfo{##1}}}}
3509 \@ifpackagewith{babel}{ensureinfo=off}{}%
3510   {\AtEndOfPackage{% Test for plain.
3511     \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.

```
3512 \newcommand\getlocaleproperty{%
3513   \@ifstar\bbl@getproperty@s\bbl@getproperty@x}
3514 \def\bbl@getproperty@s#1#2#3{%
3515   \let#1\relax
3516   \def\bbl@elt##1##2##3{%
3517     \bbl@ifsamestring{##1/##2}{#3}%
3518       {\providecommand#1{##3}%
3519        \def\bbl@elt####1####2####3{}}%
3520       {}}%
3521   \bbl@cs{inidata@#2}}%
3522 \def\bbl@getproperty@x#1#2#3{%
3523   \bbl@getproperty@s{#1}{#2}{#3}%
3524   \ifx#1\relax
3525     \bbl@error
3526       {Unknown key for locale '#2':\\%
3527        #3\\%
3528        \string#1 will be set to \relax}%
3529       {Perhaps you misspelled it.}%
3530   \fi}
3531 \let\bbl@ini@loaded\@empty
3532 \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.

```
3533 \newcommand\babeladjust[1]{%  TODO. Error handling.
3534   \bbl@forkv{#1}{%
3535     \bbl@ifunset{bbl@ADJ@##1@##2}%
3536       {\bbl@cs{ADJ@##1}{##2}}%
3537       {\bbl@cs{ADJ@##1@##2}}}}
```

```
3538 %
3539 \def\bbl@adjust@lua#1#2{%
3540   \ifvmode
3541     \ifnum\currentgrouplevel=\z@
3542       \directlua{ Babel.#2 }%
3543       \expandafter\expandafter\expandafter\@gobble
3544     \fi
3545   \fi
3546   {\bbl@error   % The error is gobbled if everything went ok.
3547     {Currently, #1 related features can be adjusted only\\%
3548      in the main vertical list.}%
3549     {Maybe things change in the future, but this is what it is.}}}
3550 \@namedef{bbl@ADJ@bidi.mirroring@on}{%
3551   \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
3552 \@namedef{bbl@ADJ@bidi.mirroring@off}{%
3553   \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
3554 \@namedef{bbl@ADJ@bidi.text@on}{%
3555   \bbl@adjust@lua{bidi}{bidi_enabled=true}}
3556 \@namedef{bbl@ADJ@bidi.text@off}{%
3557   \bbl@adjust@lua{bidi}{bidi_enabled=false}}
3558 \@namedef{bbl@ADJ@bidi.mapdigits@on}{%
3559   \bbl@adjust@lua{bidi}{digits_mapped=true}}
3560 \@namedef{bbl@ADJ@bidi.mapdigits@off}{%
3561   \bbl@adjust@lua{bidi}{digits_mapped=false}}
3562 %
3563 \@namedef{bbl@ADJ@linebreak.sea@on}{%
3564   \bbl@adjust@lua{linebreak}{sea_enabled=true}}
3565 \@namedef{bbl@ADJ@linebreak.sea@off}{%
3566   \bbl@adjust@lua{linebreak}{sea_enabled=false}}
3567 \@namedef{bbl@ADJ@linebreak.cjk@on}{%
3568   \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
3569 \@namedef{bbl@ADJ@linebreak.cjk@off}{%
3570   \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
3571 \@namedef{bbl@ADJ@justify.arabic@on}{%
3572   \bbl@adjust@lua{linebreak}{arabic.justify_enabled=true}}
3573 \@namedef{bbl@ADJ@justify.arabic@off}{%
3574   \bbl@adjust@lua{linebreak}{arabic.justify_enabled=false}}
3575 %
3576 \def\bbl@adjust@layout#1{%
3577   \ifvmode
3578     #1%
3579     \expandafter\@gobble
3580   \fi
3581   {\bbl@error   % The error is gobbled if everything went ok.
3582     {Currently, layout related features can be adjusted only\\%
3583      in vertical mode.}%
3584     {Maybe things change in the future, but this is what it is.}}}
3585 \@namedef{bbl@ADJ@layout.tabular@on}{%
3586   \bbl@adjust@layout{\let\@tabular\bbl@NL@@tabular}}
3587 \@namedef{bbl@ADJ@layout.tabular@off}{%
3588   \bbl@adjust@layout{\let\@tabular\bbl@OL@@tabular}}
3589 \@namedef{bbl@ADJ@layout.lists@on}{%
3590   \bbl@adjust@layout{\let\list\bbl@NL@list}}
3591 \@namedef{bbl@ADJ@layout.lists@off}{%
3592   \bbl@adjust@layout{\let\list\bbl@OL@list}}
3593 \@namedef{bbl@ADJ@hyphenation.extra@on}{%
3594   \bbl@activateposthyphen}
3595 %
3596 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
3597   \bbl@bcpallowedtrue}
3598 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
3599   \bbl@bcpallowedfalse}
3600 \@namedef{bbl@ADJ@autoload.bcp47.prefix}#1{%
```

```
3601    \def\bbl@bcp@prefix{#1}}
3602 \def\bbl@bcp@prefix{bcp47-}
3603 \@namedef{bbl@ADJ@autoload.options}#1{%
3604    \def\bbl@autoload@options{#1}}
3605 \let\bbl@autoload@bcpoptions\@empty
3606 \@namedef{bbl@ADJ@autoload.bcp47.options}#1{%
3607    \def\bbl@autoload@bcpoptions{#1}}
3608 \newif\ifbbl@bcptoname
3609 \@namedef{bbl@ADJ@bcp47.toname@on}{%
3610    \bbl@bcptonametrue
3611    \BabelEnsureInfo}
3612 \@namedef{bbl@ADJ@bcp47.toname@off}{%
3613    \bbl@bcptonamefalse}
3614 \@namedef{bbl@ADJ@prehyphenation.disable@nohyphenation}{%
3615    \directlua{ Babel.ignore_pre_char = function(node)
3616        return (node.lang == \the\csname l@nohyphenation\endcsname)
3617      end }}
3618 \@namedef{bbl@ADJ@prehyphenation.disable@off}{%
3619    \directlua{ Babel.ignore_pre_char = function(node)
3620        return false
3621      end }}
3622 \@namedef{bbl@ADJ@select.write@shift}{%
3623    \let\bbl@restorelastskip\relax
3624    \def\bbl@savelastskip{%
3625      \let\bbl@restorelastskip\relax
3626      \ifvmode
3627        \ifdim\lastskip=\z@
3628          \let\bbl@restorelastskip\nobreak
3629        \else
3630          \bbl@exp{%
3631            \def\\\bbl@restorelastskip{%
3632              \skip@=\the\lastskip
3633              \\\nobreak \vskip-\skip@ \vskip\skip@}}%
3634        \fi
3635      \fi}}
3636 \@namedef{bbl@ADJ@select.write@keep}{%
3637    \let\bbl@restorelastskip\relax
3638    \let\bbl@savelastskip\relax}
3639 \@namedef{bbl@ADJ@select.write@omit}{%
3640    \let\bbl@restorelastskip\relax
3641    \def\bbl@savelastskip##1\bbl@restorelastskip{}}
```

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

```
3642 \ifx\directlua\@undefined\else
3643    \ifx\bbl@luapatterns\@undefined
3644      \input luababel.def
3645    \fi
3646 \fi
```

Continue with LaTeX.

```
3647 ⟨/package | core⟩
3648 ⟨∗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.

```
3649 ⟨⟨*More package options⟩⟩ ≡
3650 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
3651 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
3652 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
3653 \DeclareOption{safe=refbib}{\def\bbl@opt@safe{BR}}
3654 \DeclareOption{safe=bibref}{\def\bbl@opt@safe{BR}}
3655 ⟨⟨/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.

```
3656 \bbl@trace{Cross referencing macros}
3657 \ifx\bbl@opt@safe\@empty\else % ie, if 'ref' and/or 'bib'
3658   \def\@newl@bel#1#2#3{%
3659    {\@safe@activestrue
3660     \bbl@ifunset{#1@#2}%
3661        \relax
3662        {\gdef\@multiplelabels{%
3663           \@latex@warning@no@line{There were multiply-defined labels}}%
3664         \@latex@warning@no@line{Label `#2' multiply defined}}%
3665     \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.

```
3666   \CheckCommand*\@testdef[3]{%
3667     \def\reserved@a{#3}%
3668     \expandafter\ifx\csname#1@#2\endcsname\reserved@a
3669     \else
3670       \@tempswatrue
3671     \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.

```
3672   \def\@testdef#1#2#3{%  TODO. With @samestring?
3673     \@safe@activestrue
3674     \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
3675     \def\bbl@tempb{#3}%
3676     \@safe@activesfalse
3677     \ifx\bbl@tempa\relax
3678     \else
3679       \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
3680     \fi
3681     \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
3682     \ifx\bbl@tempa\bbl@tempb
3683     \else
3684       \@tempswatrue
3685     \fi}
3686 \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.

```
3687 \bbl@xin@{R}\bbl@opt@safe
3688 \ifin@
3689   \edef\bbl@tempc{\expandafter\string\csname ref code\endcsname}%
3690   \bbl@xin@{\expandafter\strip@prefix\meaning\bbl@tempc}%
3691     {\expandafter\strip@prefix\meaning\ref}%
3692   \ifin@
```

```
3693    \bbl@redefine\@kernel@ref#1{%
3694      \@safe@activestrue\org@@kernel@ref{#1}\@safe@activesfalse}
3695    \bbl@redefine\@kernel@pageref#1{%
3696      \@safe@activestrue\org@@kernel@pageref{#1}\@safe@activesfalse}
3697    \bbl@redefine\@kernel@sref#1{%
3698      \@safe@activestrue\org@@kernel@sref{#1}\@safe@activesfalse}
3699    \bbl@redefine\@kernel@spageref#1{%
3700      \@safe@activestrue\org@@kernel@spageref{#1}\@safe@activesfalse}
3701  \else
3702    \bbl@redefinerobust\ref#1{%
3703      \@safe@activestrue\org@ref{#1}\@safe@activesfalse}
3704    \bbl@redefinerobust\pageref#1{%
3705      \@safe@activestrue\org@pageref{#1}\@safe@activesfalse}
3706  \fi
3707 \else
3708  \let\org@ref\ref
3709  \let\org@pageref\pageref
3710 \fi
```

**\@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.

```
3711 \bbl@xin@{B}\bbl@opt@safe
3712 \ifin@
3713  \bbl@redefine\@citex[#1]#2{%
3714    \@safe@activestrue\edef\@tempa{#2}\@safe@activesfalse
3715    \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.

```
3716    \AtBeginDocument{%
3717      \@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.)

```
3718      \def\@citex[#1][#2]#3{%
3719        \@safe@activestrue\edef\@tempa{#3}\@safe@activesfalse
3720        \org@@citex[#1][#2]{\@tempa}}%
3721      }{}}
```

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

```
3722    \AtBeginDocument{%
3723      \@ifpackageloaded{cite}{%
3724        \def\@citex[#1]#2{%
3725          \@safe@activestrue\org@@citex[#1]{#2}\@safe@activesfalse}%
3726        }{}}
```

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

```
3727 \bbl@redefine\nocite#1{%
3728    \@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.

```
3729  \bbl@redefine\bibcite{%
3730    \bbl@cite@choice
3731    \bibcite}
```

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

```
3732  \def\bbl@bibcite#1#2{%
3733    \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.

```
3734  \def\bbl@cite@choice{%
3735    \global\let\bibcite\bbl@bibcite
3736    \@ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{}%
3737    \@ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{}%
3738    \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.

```
3739  \AtBeginDocument{\bbl@cite@choice}
```

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

```
3740  \bbl@redefine\@bibitem#1{%
3741    \@safe@activestrue\org@@bibitem{#1}\@safe@activesfalse}
3742 \else
3743    \let\org@nocite\nocite
3744    \let\org@@citex\@citex
3745    \let\org@bibcite\bibcite
3746    \let\org@@bibitem\@bibitem
3747 \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.

```
3748 \bbl@trace{Marks}
3749 \IfBabelLayout{sectioning}
3750   {\ifx\bbl@opt@headfoot\@nnil
3751     \g@addto@macro\@resetactivechars{%
3752       \set@typeset@protect
3753       \expandafter\select@language@x\expandafter{\bbl@main@language}%
3754       \let\protect\noexpand
3755       \ifcase\bbl@bidimode\else % Only with bidi. See also above
3756         \edef\thepage{%
3757           \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
3758       \fi}%
3759   \fi}
3760   {\ifbbl@single\else
3761     \bbl@ifunset{markright }\bbl@redefine\bbl@redefinerobust
3762     \markright#1{%
3763       \bbl@ifblank{#1}%
3764         {\org@markright{}}%
3765         {\toks@{#1}%
3766          \bbl@exp{%
3767            \\\org@markright{\\\protect\\\foreignlanguage{\languagename}%
3768              {\\\protect\\\bbl@restore@actives\the\toks@}}}}}%
```

|  | \markboth | The definition of \markboth is equivalent to that of \markright, except that we need two token |
| \markboth | | registers. The documentclasses report and book define and set the headings for the page. While |
| \@mkboth | | doing so they also store a copy of \markboth in \@mkboth. Therefore we need to check whether |

\markboth
\@mkboth
The definition of \markboth is equivalent to that of \markright, except that we need two token registers. The documentclasses report and book define and set the headings for the page. While doing so they also store a copy of \markboth in \@mkboth. Therefore we need to check whether \@mkboth has already been set. If so we 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.)

```
3769    \ifx\@mkboth\markboth
3770      \def\bbl@tempc{\let\@mkboth\markboth}
3771    \else
3772      \def\bbl@tempc{}
3773    \fi
3774    \bbl@ifunset{markboth }\bbl@redefine\bbl@redefinerobust
3775    \markboth#1#2{%
3776      \protected@edef\bbl@tempb##1{%
3777        \protect\foreignlanguage
3778        {\languagename}{\protect\bbl@restore@actives##1}}%
3779      \bbl@ifblank{#1}%
3780        {\toks@{}}%
3781        {\toks@\expandafter{\bbl@tempb{#1}}}%
3782      \bbl@ifblank{#2}%
3783        {\@temptokena{}}%
3784        {\@temptokena\expandafter{\bbl@tempb{#2}}}%
3785      \bbl@exp{\\\org@markboth{\the\toks@}{\the\@temptokena}}}
3786      \bbl@tempc
3787    \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.

```
3788 \bbl@trace{Preventing clashes with other packages}
3789 \ifx\org@ref\@undefined\else
3790   \bbl@xin@{R}\bbl@opt@safe
3791   \ifin@
3792     \AtBeginDocument{%
3793       \@ifpackageloaded{ifthen}{%
3794         \bbl@redefine@long\ifthenelse#1#2#3{%
3795           \let\bbl@temp@pref\pageref
3796           \let\pageref\org@pageref
3797           \let\bbl@temp@ref\ref
3798           \let\ref\org@ref
3799           \@safe@activestrue
3800           \org@ifthenelse{#1}%
3801             {\let\pageref\bbl@temp@pref
3802              \let\ref\bbl@temp@ref
3803              \@safe@activesfalse
3804              #2}%
```

```
3805          {\let\pageref\bbl@temp@pref
3806           \let\ref\bbl@temp@ref
3807           \@safe@activesfalse
3808           #3}%
3809        }%
3810      }{}%
3811    }
3812 \fi
```

### 8.3.2 varioref

\@@vpageref  
\vrefpagenum  
\Ref

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

```
3813    \AtBeginDocument{%
3814      \@ifpackageloaded{varioref}{%
3815        \bbl@redefine\@@vpageref#1[#2]#3{%
3816          \@safe@activestrue
3817          \org@@@vpageref{#1}[#2]{#3}%
3818          \@safe@activesfalse}%
3819        \bbl@redefine\vrefpagenum#1#2{%
3820          \@safe@activestrue
3821          \org@vrefpagenum{#1}{#2}%
3822          \@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.

```
3823        \expandafter\def\csname Ref \endcsname#1{%
3824          \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
3825      }{}%
3826    }
3827 \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.

```
3828 \AtEndOfPackage{%
3829    \AtBeginDocument{%
3830      \@ifpackageloaded{hhline}%
3831        {\expandafter\ifx\csname normal@char\string:\endcsname\relax
3832         \else
3833           \makeatletter
3834           \def\@currname{hhline}\input{hhline.sty}\makeatother
3835        \fi}%
3836        {}}}
```

\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.

```
3837 \def\substitutefontfamily#1#2#3{%
3838    \lowercase{\immediate\openout15=#1#2.fd\relax}%
3839    \immediate\write15{%
3840      \string\ProvidesFile{#1#2.fd}%
3841      [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
3842       \space generated font description file]^^J
```

```
3843      \string\DeclareFontFamily{#1}{#2}{}{}^^J
3844      \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{}^^J
3845      \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{}^^J
3846      \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{}^^J
3847      \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{}^^J
3848      \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{}^^J
3849      \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{}^^J
3850      \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{}^^J
3851      \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{}^^J
3852      }%
3853    \closeout15
3854    }
3855 \@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

```
3856 \bbl@trace{Encoding and fonts}
3857 \newcommand\BabelNonASCII{LGR,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU}
3858 \newcommand\BabelNonText{TS1,T3,TS3}
3859 \let\org@TeX\TeX
3860 \let\org@LaTeX\LaTeX
3861 \let\ensureascii\@firstofone
3862 \AtBeginDocument{%
3863   \def\@elt#1{,#1,}%
3864   \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3865   \let\@elt\relax
3866   \let\bbl@tempb\@empty
3867   \def\bbl@tempc{OT1}%
3868   \bbl@foreach\BabelNonASCII{% LGR loaded in a non-standard way
3869     \bbl@ifunset{T@#1}{}{\def\bbl@tempb{#1}}}%
3870   \bbl@foreach\bbl@tempa{%
3871     \bbl@xin@{#1}{\BabelNonASCII}%
3872     \ifin@
3873       \def\bbl@tempb{#1}% Store last non-ascii
3874     \else\bbl@xin@{#1}{\BabelNonText}% Pass
3875       \ifin@\else
3876         \def\bbl@tempc{#1}% Store last ascii
3877       \fi
3878     \fi}%
3879   \ifx\bbl@tempb\@empty\else
3880     \bbl@xin@{,\cf@encoding,}{,\BabelNonASCII,\BabelNonText,}%
3881     \ifin@\else
3882       \edef\bbl@tempc{\cf@encoding}% The default if ascii wins
3883     \fi
3884     \edef\ensureascii#1{%
3885       {\noexpand\fontencoding{\bbl@tempc}\noexpand\selectfont#1}}%
3886     \DeclareTextCommandDefault{\TeX}{\ensureascii{\org@TeX}}%
3887     \DeclareTextCommandDefault{\LaTeX}{\ensureascii{\org@LaTeX}}%
3888   \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.

```
3889 \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.

```
3890 \AtBeginDocument{%
3891   \@ifpackageloaded{fontspec}%
3892     {\xdef\latinencoding{%
3893        \ifx\UTFencname\@undefined
3894          EU\ifcase\bbl@engine\or2\or1\fi
3895        \else
3896          \UTFencname
3897        \fi}}%
3898     {\gdef\latinencoding{OT1}%
3899      \ifx\cf@encoding\bbl@t@one
3900        \xdef\latinencoding{\bbl@t@one}%
3901      \else
3902        \def\@elt#1{,#1,}%
3903        \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3904        \let\@elt\relax
3905        \bbl@xin@{,T1,}\bbl@tempa
3906        \ifin@
3907          \xdef\latinencoding{\bbl@t@one}%
3908        \fi
3909      \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.

```
3910 \DeclareRobustCommand{\latintext}{%
3911   \fontencoding{\latinencoding}\selectfont
3912   \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.

```
3913 \ifx\@undefined\DeclareTextFontCommand
3914   \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
3915 \else
3916   \DeclareTextFontCommand{\textlatin}{\latintext}
3917 \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).

```
3918 \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.

```
3919 \bbl@trace{Loading basic (internal) bidi support}
3920 \ifodd\bbl@engine
3921 \else % TODO. Move to txtbabel
3922   \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
3923     \bbl@error
3924       {The bidi method 'basic' is available only in\\%
3925        luatex. I'll continue with 'bidi=default', so\\%
3926        expect wrong results}%
3927       {See the manual for further details.}%
3928     \let\bbl@beforeforeign\leavevmode
3929     \AtEndOfPackage{%
3930       \EnableBabelHook{babel-bidi}%
3931       \bbl@xebidipar}
3932   \fi\fi
3933   \def\bbl@loadxebidi#1{%
3934     \ifx\RTLfootnotetext\@undefined
3935       \AtEndOfPackage{%
3936         \EnableBabelHook{babel-bidi}%
3937         \ifx\fontspec\@undefined
3938           \bbl@loadfontspec % bidi needs fontspec
3939         \fi
3940         \usepackage#1{bidi}}%
3941     \fi}
3942   \ifnum\bbl@bidimode>200
3943     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
3944       \bbl@tentative{bidi=bidi}
3945       \bbl@loadxebidi{}
3946     \or
3947       \bbl@loadxebidi{[rldocument]}
3948     \or
3949       \bbl@loadxebidi{}
3950     \fi
3951   \fi
3952 \fi
3953 % TODO? Separate:
3954 \ifnum\bbl@bidimode=\@ne
3955   \let\bbl@beforeforeign\leavevmode
3956   \ifodd\bbl@engine
3957     \newattribute\bbl@attr@dir
3958     \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
3959     \bbl@exp{\output{\bodydir\pagedir\the\output}}
3960   \fi
3961   \AtEndOfPackage{%
3962     \EnableBabelHook{babel-bidi}%
3963     \ifodd\bbl@engine\else
3964       \bbl@xebidipar
3965     \fi}
3966 \fi
```

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

```
3967 \bbl@trace{Macros to switch the text direction}
3968 \def\bbl@alscripts{,Arabic,Syriac,Thaana,}
3969 \def\bbl@rscripts{% TODO. Base on codes ??
3970   ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
3971   Old Hungarian,Old Hungarian,Lydian,Mandaean,Manichaean,%
3972   Manichaean,Meroitic Cursive,Meroitic,Old North Arabian,%
3973   Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
3974   Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
```

```
3975    Old South Arabian,}%
3976 \def\bbl@provide@dirs#1{%
3977    \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
3978    \ifin@
3979      \global\bbl@csarg\chardef{wdir@#1}\@ne
3980      \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
3981      \ifin@
3982        \global\bbl@csarg\chardef{wdir@#1}\tw@  % useless in xetex
3983      \fi
3984    \else
3985      \global\bbl@csarg\chardef{wdir@#1}\z@
3986    \fi
3987    \ifodd\bbl@engine
3988      \bbl@csarg\ifcase{wdir@#1}%
3989        \directlua{ Babel.locale_props[\the\localeid].textdir = 'l' }%
3990      \or
3991        \directlua{ Babel.locale_props[\the\localeid].textdir = 'r' }%
3992      \or
3993        \directlua{ Babel.locale_props[\the\localeid].textdir = 'al' }%
3994      \fi
3995    \fi}
3996 \def\bbl@switchdir{%
3997    \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
3998    \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
3999    \bbl@exp{\\\bbl@setdirs\bbl@cl{wdir}}}
4000 \def\bbl@setdirs#1{% TODO - math
4001    \ifcase\bbl@select@type % TODO - strictly, not the right test
4002      \bbl@bodydir{#1}%
4003      \bbl@pardir{#1}%
4004    \fi
4005    \bbl@textdir{#1}}
4006 % TODO. Only if \bbl@bidimode > 0?:
4007 \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
4008 \DisableBabelHook{babel-bidi}
```

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

```
4009 \ifodd\bbl@engine  % luatex=1
4010 \else % pdftex=0, xetex=2
4011    \newcount\bbl@dirlevel
4012    \chardef\bbl@thetextdir\z@
4013    \chardef\bbl@thepardir\z@
4014    \def\bbl@textdir#1{%
4015      \ifcase#1\relax
4016        \chardef\bbl@thetextdir\z@
4017        \bbl@textdir@i\beginL\endL
4018      \else
4019        \chardef\bbl@thetextdir\@ne
4020        \bbl@textdir@i\beginR\endR
4021      \fi}
4022    \def\bbl@textdir@i#1#2{%
4023      \ifhmode
4024        \ifnum\currentgrouplevel>\z@
4025          \ifnum\currentgrouplevel=\bbl@dirlevel
4026            \bbl@error{Multiple bidi settings inside a group}%
4027              {I'll insert a new group, but expect wrong results.}%
4028            \bgroup\aftergroup#2\aftergroup\egroup
4029          \else
4030            \ifcase\currentgrouptype\or % 0 bottom
4031              \aftergroup#2% 1 simple {}
4032            \or
4033              \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
4034            \or
4035              \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
```

```
4036          \or\or\or % vbox vtop align
4037          \or
4038            \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
4039          \or\or\or\or\or\or % output math disc insert vcent mathchoice
4040          \or
4041            \aftergroup#2% 14 \begingroup
4042          \else
4043            \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
4044          \fi
4045        \fi
4046        \bbl@dirlevel\currentgrouplevel
4047      \fi
4048      #1%
4049    \fi}
4050  \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
4051  \let\bbl@bodydir\@gobble
4052  \let\bbl@pagedir\@gobble
4053  \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).

```
4054  \def\bbl@xebidipar{%
4055    \let\bbl@xebidipar\relax
4056    \TeXXeTstate\@ne
4057    \def\bbl@xeeverypar{%
4058      \ifcase\bbl@thepardir
4059        \ifcase\bbl@thetextdir\else\beginR\fi
4060      \else
4061        {\setbox\z@\lastbox\beginR\box\z@}%
4062      \fi}%
4063    \let\bbl@severypar\everypar
4064    \newtoks\everypar
4065    \everypar=\bbl@severypar
4066    \bbl@severypar{\bbl@xeeverypar\the\everypar}}
4067  \ifnum\bbl@bidimode>200
4068    \let\bbl@textdir@i\@gobbletwo
4069    \let\bbl@xebidipar\@empty
4070    \AddBabelHook{bidi}{foreign}{%
4071      \def\bbl@tempa{\def\BabelText####1}%
4072      \ifcase\bbl@thetextdir
4073        \expandafter\bbl@tempa\expandafter{\BabelText{\LR{##1}}}%
4074      \else
4075        \expandafter\bbl@tempa\expandafter{\BabelText{\RL{##1}}}%
4076      \fi}
4077    \def\bbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}
4078  \fi
4079 \fi
```

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

```
4080 \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}
4081 \AtBeginDocument{%
4082  \ifx\pdfstringdefDisableCommands\@undefined\else
4083    \ifx\pdfstringdefDisableCommands\relax\else
4084      \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
4085    \fi
4086  \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`.

```
4087 \bbl@trace{Local Language Configuration}
4088 \ifx\loadlocalcfg\@undefined
4089   \@ifpackagewith{babel}{noconfigs}%
4090     {\let\loadlocalcfg\@gobble}%
4091     {\def\loadlocalcfg#1{%
4092       \InputIfFileExists{#1.cfg}%
4093         {\typeout{*************************************^^J%
4094                       * Local config file #1.cfg used^^J%
4095                       *}}%
4096       \@empty}}
4097 \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).

```
4098 \bbl@trace{Language options}
4099 \let\bbl@afterlang\relax
4100 \let\BabelModifiers\relax
4101 \let\bbl@loaded\@empty
4102 \def\bbl@load@language#1{%
4103   \InputIfFileExists{#1.ldf}%
4104     {\edef\bbl@loaded{\CurrentOption
4105       \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}%
4106     \expandafter\let\expandafter\bbl@afterlang
4107       \csname\CurrentOption.ldf-h@@k\endcsname
4108     \expandafter\let\expandafter\BabelModifiers
4109       \csname bbl@mod@\CurrentOption\endcsname}%
4110     {\bbl@error{%
4111       Unknown option '\CurrentOption'. Either you misspelled it\\%
4112       or the language definition file \CurrentOption.ldf was not found}{%
4113       Valid options are, among others: shorthands=, KeepShorthandsActive,\\%
4114       activeacute, activegrave, noconfigs, safe=, main=, math=\\%
4115       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.

```
4116 \def\bbl@try@load@lang#1#2#3{%
4117   \IfFileExists{\CurrentOption.ldf}%
4118     {\bbl@load@language{\CurrentOption}}%
4119     {#1\bbl@load@language{#2}#3}}
4120 %
4121 \DeclareOption{hebrew}{%
4122   \input{rlbabel.def}%
4123   \bbl@load@language{hebrew}}
4124 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}{}}
4125 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}{}}
4126 \DeclareOption{nynorsk}{\bbl@try@load@lang{}{norsk}{}}
4127 \DeclareOption{polutonikogreek}{%
4128   \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}
4129 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}{}}
4130 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}{}}
4131 \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.

```
4132 \ifx\bbl@opt@config\@nnil
4133   \@ifpackagewith{babel}{noconfigs}{}%
```

```
4134    {\InputIfFileExists{bblopts.cfg}%
4135      {\typeout{************************************^^J%
4136                * Local config file bblopts.cfg used^^J%
4137                *}}%
4138      {}}%
4139 \else
4140   \InputIfFileExists{\bbl@opt@config.cfg}%
4141    {\typeout{************************************^^J%
4142                * Local config file \bbl@opt@config.cfg used^^J%
4143                *}}%
4144    {\bbl@error{%
4145        Local config file '\bbl@opt@config.cfg' not found}{%
4146        Perhaps you misspelled it.}}%
4147 \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.

```
4148 \ifx\bbl@opt@main\@nnil
4149   \ifnum\bbl@iniflag>\z@  % if all ldf's: set implicitly, no main pass
4150     \let\bbl@tempb\@empty
4151     \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}%
4152     \bbl@foreach\bbl@tempa{\edef\bbl@tempb{#1,\bbl@tempb}}%
4153     \bbl@foreach\bbl@tempb{%     \bbl@tempb is a reversed list
4154       \ifx\bbl@opt@main\@nnil % ie, if not yet assigned
4155         \ifodd\bbl@iniflag % = *=
4156           \IfFileExists{babel-#1.tex}{\def\bbl@opt@main{#1}}{}%
4157         \else % n +=
4158           \IfFileExists{#1.ldf}{\def\bbl@opt@main{#1}}{}%
4159         \fi
4160       \fi}%
4161   \fi
4162 \else
4163   \bbl@info{Main language set with 'main='. Except if you have\\%
4164             problems, prefer the default mechanism for setting\\%
4165             the main language. Reported}
4166 \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).

```
4167 \ifx\bbl@opt@main\@nnil\else
4168   \bbl@csarg\let{loadmain\expandafter}\csname ds@\bbl@opt@main\endcsname
4169   \expandafter\let\csname ds@\bbl@opt@main\endcsname\relax
4170 \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.

```
4171 \bbl@foreach\bbl@language@opts{%
4172   \def\bbl@tempa{#1}%
4173   \ifx\bbl@tempa\bbl@opt@main\else
4174     \ifnum\bbl@iniflag<\tw@    % 0 ø (other = ldf)
4175       \bbl@ifunset{ds@#1}%
4176         {\DeclareOption{#1}{\bbl@load@language{#1}}}%
4177         {}%
4178     \else                      % + * (other = ini)
4179       \DeclareOption{#1}{%
4180         \bbl@ldfinit
4181         \babelprovide[import]{#1}%
4182         \bbl@afterldf{}}%
4183     \fi
```

```
4184    \fi}
4185 \bbl@foreach\@classoptionslist{%
4186    \def\bbl@tempa{#1}%
4187    \ifx\bbl@tempa\bbl@opt@main\else
4188      \ifnum\bbl@iniflag<\tw@     % 0 ø (other = ldf)
4189        \bbl@ifunset{ds@#1}%
4190          {\IfFileExists{#1.ldf}%
4191            {\DeclareOption{#1}{\bbl@load@language{#1}}}%
4192            {}}%
4193          {}%
4194      \else                       % + * (other = ini)
4195        \IfFileExists{babel-#1.tex}%
4196          {\DeclareOption{#1}{%
4197              \bbl@ldfinit
4198              \babelprovide[import]{#1}%
4199              \bbl@afterldf{}}}%
4200          {}%
4201      \fi
4202    \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):

```
4203 \def\AfterBabelLanguage#1{%
4204    \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang}{}}
4205 \DeclareOption*{}
4206 \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.

```
4207 \bbl@trace{Option 'main'}
4208 \ifx\bbl@opt@main\@nnil
4209    \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}
4210    \let\bbl@tempc\@empty
4211    \bbl@for\bbl@tempb\bbl@tempa{%
4212      \bbl@xin@{,\bbl@tempb,}{,\bbl@loaded,}%
4213      \ifin@\edef\bbl@tempc{\bbl@tempb}\fi}
4214    \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
4215    \expandafter\bbl@tempa\bbl@loaded,\@nnil
4216    \ifx\bbl@tempb\bbl@tempc\else
4217      \bbl@warning{%
4218        Last declared language option is '\bbl@tempc',\\%
4219        but the last processed one was '\bbl@tempb'.\\%
4220        The main language can't be set as both a global\\%
4221        and a package option. Use 'main=\bbl@tempc' as\\%
4222        option. Reported}
4223    \fi
4224 \else
4225    \ifodd\bbl@iniflag  % case 1,3 (main is ini)
4226      \bbl@ldfinit
4227      \let\CurrentOption\bbl@opt@main
4228      \bbl@exp{%  \bbl@opt@provide = empty if *
4229          \\\babelprovide[\bbl@opt@provide,import,main]{\bbl@opt@main}}%
4230      \bbl@afterldf{}
4231      \DeclareOption{\bbl@opt@main}{}
4232    \else % case 0,2 (main is ldf)
4233      \ifx\bbl@loadmain\relax
4234        \DeclareOption{\bbl@opt@main}{\bbl@load@language{\bbl@opt@main}}
4235      \else
```

148

```
4236        \DeclareOption{\bbl@opt@main}{\bbl@loadmain}
4237      \fi
4238      \ExecuteOptions{\bbl@opt@main}
4239      \@namedef{ds@\bbl@opt@main}{}%
4240    \fi
4241    \DeclareOption*{}
4242    \ProcessOptions*
4243 \fi
4244 \def\AfterBabelLanguage{%
4245    \bbl@error
4246      {Too late for \string\AfterBabelLanguage}%
4247      {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.

```
4248 \ifx\bbl@main@language\@undefined
4249    \bbl@info{%
4250      You haven't specified a language. I'll use 'nil'\\%
4251      as the main language. Reported}
4252      \bbl@load@language{nil}
4253 \fi
4254 ⟨/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

```
4255 ⟨*kernel⟩
4256 \let\bbl@onlyswitch\@empty
4257 \input babel.def
4258 \let\bbl@onlyswitch\@undefined
4259 ⟨/kernel⟩
4260 ⟨*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.

```
4261 ⟨⟨Make sure ProvidesFile is defined⟩⟩
4262 \ProvidesFile{hyphen.cfg}[⟨⟨date⟩⟩ ⟨⟨version⟩⟩ Babel hyphens]
4263 \xdef\bbl@format{\jobname}
4264 \def\bbl@version{⟨⟨version⟩⟩}
4265 \def\bbl@date{⟨⟨date⟩⟩}
4266 \ifx\AtBeginDocument\@undefined
4267    \def\@empty{}
4268 \fi
4269 ⟨⟨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.

```
4270 \def\process@line#1#2 #3 #4 {%
4271   \ifx=#1%
4272     \process@synonym{#2}%
4273   \else
4274     \process@language{#1#2}{#3}{#4}%
4275   \fi
4276   \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.

```
4277 \toks@{}
4278 \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.

```
4279 \def\process@synonym#1{%
4280   \ifnum\last@language=\m@ne
4281     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4282   \else
4283     \expandafter\chardef\csname l@#1\endcsname\last@language
4284     \wlog{\string\l@#1=\string\language\the\last@language}%
4285     \expandafter\let\csname #1hyphenmins\expandafter\endcsname
4286       \csname\languagename hyphenmins\endcsname
4287     \let\bbl@elt\relax
4288     \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}{}{}}%
4289   \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.

```
4290 \def\process@language#1#2#3{%
4291   \expandafter\addlanguage\csname l@#1\endcsname
4292   \expandafter\language\csname l@#1\endcsname
4293   \edef\languagename{#1}%
4294   \bbl@hook@everylanguage{#1}%
4295   % > luatex
```

```
4296    \bbl@get@enc#1::\@@@
4297    \begingroup
4298      \lefthyphenmin\m@ne
4299      \bbl@hook@loadpatterns{#2}%
4300      %  > luatex
4301      \ifnum\lefthyphenmin=\m@ne
4302      \else
4303        \expandafter\xdef\csname #1hyphenmins\endcsname{%
4304          \the\lefthyphenmin\the\righthyphenmin}%
4305      \fi
4306    \endgroup
4307    \def\bbl@tempa{#3}%
4308    \ifx\bbl@tempa\@empty\else
4309      \bbl@hook@loadexceptions{#3}%
4310      %  > luatex
4311    \fi
4312    \let\bbl@elt\relax
4313    \edef\bbl@languages{%
4314      \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
4315    \ifnum\the\language=\z@
4316      \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4317        \set@hyphenmins\tw@\thr@@\relax
4318      \else
4319        \expandafter\expandafter\expandafter\set@hyphenmins
4320          \csname #1hyphenmins\endcsname
4321      \fi
4322      \the\toks@
4323      \toks@{}%
4324    \fi}
```

\bbl@get@enc
\bbl@hyph@enc  The macro \bbl@get@enc extracts the font encoding from the language name and stores it in
\bbl@hyph@enc. It uses delimited arguments to achieve this.

```
4325 \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.

```
4326 \def\bbl@hook@everylanguage#1{}
4327 \def\bbl@hook@loadpatterns#1{\input #1\relax}
4328 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
4329 \def\bbl@hook@loadkernel#1{%
4330    \def\addlanguage{\csname newlanguage\endcsname}%
4331    \def\adddialect##1##2{%
4332      \global\chardef##1##2\relax
4333      \wlog{\string##1 = a dialect from \string\language##2}}%
4334    \def\iflanguage##1{%
4335      \expandafter\ifx\csname l@##1\endcsname\relax
4336        \@nolanerr{##1}%
4337      \else
4338        \ifnum\csname l@##1\endcsname=\language
4339          \expandafter\expandafter\expandafter\@firstoftwo
4340        \else
4341          \expandafter\expandafter\expandafter\@secondoftwo
4342        \fi
4343      \fi}%
4344    \def\providehyphenmins##1##2{%
4345      \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
4346        \@namedef{##1hyphenmins}{##2}%
4347      \fi}%
4348    \def\set@hyphenmins##1##2{%
4349      \lefthyphenmin##1\relax
4350      \righthyphenmin##2\relax}%
4351    \def\selectlanguage{%
```

```
4352      \errhelp{Selecting a language requires a package supporting it}%
4353      \errmessage{Not loaded}}%
4354   \let\foreignlanguage\selectlanguage
4355   \let\otherlanguage\selectlanguage
4356   \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4357   \def\bbl@usehooks##1##2{}% TODO. Temporary!!
4358   \def\setlocale{%
4359      \errhelp{Find an armchair, sit down and wait}%
4360      \errmessage{Not yet available}}%
4361   \let\uselocale\setlocale
4362   \let\locale\setlocale
4363   \let\selectlocale\setlocale
4364   \let\localename\setlocale
4365   \let\textlocale\setlocale
4366   \let\textlanguage\setlocale
4367   \let\languagetext\setlocale}
4368 \begingroup
4369   \def\AddBabelHook#1#2{%
4370      \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
4371         \def\next{\toks1}%
4372      \else
4373         \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname####1}%
4374      \fi
4375      \next}
4376   \ifx\directlua\@undefined
4377      \ifx\XeTeXinputencoding\@undefined\else
4378         \input xebabel.def
4379      \fi
4380   \else
4381      \input luababel.def
4382   \fi
4383   \openin1 = babel-\bbl@format.cfg
4384   \ifeof1
4385   \else
4386      \input babel-\bbl@format.cfg\relax
4387   \fi
4388   \closein1
4389 \endgroup
4390 \bbl@hook@loadkernel{switch.def}
```

\readconfigfile   The configuration file can now be opened for reading.

```
4391 \openin1 = language.dat
```

See if the file exists, if not, use the default hyphenation file hyphen.tex. The user will be informed about this.

```
4392 \def\languagename{english}%
4393 \ifeof1
4394   \message{I couldn't find the file language.dat,\space
4395           I will try the file hyphen.tex}
4396   \input hyphen.tex\relax
4397   \chardef\l@english\z@
4398 \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.

```
4399   \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.

```
4400   \loop
```

```
4401    \endlinechar\m@ne
4402    \read1 to \bbl@line
4403    \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.

```
4404    \if T\ifeof1F\fi T\relax
4405      \ifx\bbl@line\@empty\else
4406        \edef\bbl@line{\bbl@line\space\space\space}%
4407        \expandafter\process@line\bbl@line\relax
4408      \fi
4409   \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.

```
4410    \begingroup
4411      \def\bbl@elt#1#2#3#4{%
4412        \global\language=#2\relax
4413        \gdef\languagename{#1}%
4414        \def\bbl@elt##1##2##3##4{}}%
4415      \bbl@languages
4416    \endgroup
4417 \fi
4418 \closein1
```

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

```
4419 \if/\the\toks@/\else
4420   \errhelp{language.dat loads no language, only synonyms}
4421   \errmessage{Orphan language synonym}
4422 \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.

```
4423 \let\bbl@line\@undefined
4424 \let\process@line\@undefined
4425 \let\process@synonym\@undefined
4426 \let\process@language\@undefined
4427 \let\bbl@get@enc\@undefined
4428 \let\bbl@hyph@enc\@undefined
4429 \let\bbl@tempa\@undefined
4430 \let\bbl@hook@loadkernel\@undefined
4431 \let\bbl@hook@everylanguage\@undefined
4432 \let\bbl@hook@loadpatterns\@undefined
4433 \let\bbl@hook@loadexceptions\@undefined
4434 ⟨/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].

```
4435 ⟨⟨*More package options⟩⟩ ≡
4436 \chardef\bbl@bidimode\z@
4437 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4438 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
4439 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4440 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4441 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4442 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4443 ⟨⟨/More package options⟩⟩
```

153

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.

```
4444 ⟨⟨∗Font selection⟩⟩ ≡
4445 \bbl@trace{Font handling with fontspec}
4446 \ifx\ExplSyntaxOn\@undefined\else
4447   \ExplSyntaxOn
4448   \catcode`\ =10
4449   \def\bbl@loadfontspec{%
4450     \usepackage{fontspec}%  TODO. Apply patch always
4451     \expandafter
4452     \def\csname msg~text~>~fontspec/language-not-exist\endcsname##1##2##3##4{%
4453       Font '\l_fontspec_fontname_tl' is using the\\%
4454       default features for language '##1'.\\%
4455       That's usually fine, because many languages\\%
4456       require no specific features, but if the output is\\%
4457       not as expected, consider selecting another font.}
4458     \expandafter
4459     \def\csname msg~text~>~fontspec/no-script\endcsname##1##2##3##4{%
4460       Font '\l_fontspec_fontname_tl' is using the\\%
4461       default features for script '##2'.\\%
4462       That's not always wrong, but if the output is\\%
4463       not as expected, consider selecting another font.}}
4464   \ExplSyntaxOff
4465 \fi
4466 \@onlypreamble\babelfont
4467 \newcommand\babelfont[2][]{%  1=langs/scripts 2=fam
4468   \bbl@foreach{#1}{%
4469     \expandafter\ifx\csname date##1\endcsname\relax
4470       \IfFileExists{babel-##1.tex}%
4471         {\babelprovide{##1}}%
4472         {}%
4473     \fi}%
4474   \edef\bbl@tempa{#1}%
4475   \def\bbl@tempb{#2}%  Used by \bbl@bblfont
4476   \ifx\fontspec\@undefined
4477     \bbl@loadfontspec
4478   \fi
4479   \EnableBabelHook{babel-fontspec}% Just calls \bbl@switchfont
4480   \bbl@bblfont}
4481 \newcommand\bbl@bblfont[2][]{% 1=features 2=fontname, @font=rm|sf|tt
4482   \bbl@ifunset{\bbl@tempb family}%
4483     {\bbl@providefam{\bbl@tempb}}%
4484     {}%
4485   % For the default font, just in case:
4486   \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
4487   \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
4488     {\bbl@csarg\edef{\bbl@tempb dflt@}{<>{#1}{#2}}% save bbl@rmdflt@
4489      \bbl@exp{%
4490        \let\<bbl@\bbl@tempb dflt@\languagename>\<bbl@\bbl@tempb dflt@>%
4491        \\\bbl@font@set\<bbl@\bbl@tempb dflt@\languagename>%
4492                      \<\bbl@tempb default>\<\bbl@tempb family>}}%
4493     {\bbl@foreach\bbl@tempa{% ie bbl@rmdflt@lang / *scrt
4494        \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:

```
4495 \def\bbl@providefam#1{%
4496   \bbl@exp{%
4497     \\\newcommand\<#1default>{}% Just define it
4498     \\\bbl@add@list\\\bbl@font@fams{#1}%
```

```
4499     \\\DeclareRobustCommand\<#1family>{%
4500       \\\not@math@alphabet\<#1family>\relax
4501       % \\\prepare@family@series@update{#1}\<#1default>% TODO. Fails
4502       \\\fontfamily\<#1default>%
4503       \<ifx>\\\UseHooks\\\@undefined<\else>\\\UseHook{#1family}\<fi>%
4504       \\\selectfont}%
4505     \\\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.

```
4506 \def\bbl@nostdfont#1{%
4507   \bbl@ifunset{bbl@WFF@\f@family}%
4508     {\bbl@csarg\gdef{WFF@\f@family}{}%  Flag, to avoid dupl warns
4509      \bbl@infowarn{The current font is not a babel standard family:\\%
4510        #1%
4511        \fontname\font\\%
4512        There is nothing intrinsically wrong with this warning, and\\%
4513        you can ignore it altogether if you do not need these\\%
4514        families. But if they are used in the document, you should be\\%
4515        aware 'babel' will no set Script and Language for them, so\\%
4516        you may consider defining a new family with \string\babelfont.\\%
4517        See the manual for further details about \string\babelfont.\\%
4518        Reported}}
4519   {}}%
4520 \gdef\bbl@switchfont{%
4521   \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
4522   \bbl@exp{%  eg Arabic -> arabic
4523     \lowercase{\edef\\\bbl@tempa{\bbl@cl{sname}}}}%
4524   \bbl@foreach\bbl@font@fams{%
4525     \bbl@ifunset{bbl@##1dflt@\languagename}%       (1) language?
4526       {\bbl@ifunset{bbl@##1dflt@*\bbl@tempa}%       (2) from script?
4527         {\bbl@ifunset{bbl@##1dflt@}%               2=F - (3) from generic?
4528           {}%                                      123=F - nothing!
4529           {\bbl@exp{%                              3=T - from generic
4530             \global\let\<bbl@##1dflt@\languagename>%
4531                      \<bbl@##1dflt@>}}}%
4532         {\bbl@exp{%                                2=T - from script
4533           \global\let\<bbl@##1dflt@\languagename>%
4534                      \<bbl@##1dflt@*\bbl@tempa>}}}%
4535       {}}%                                        1=T - language, already defined
4536   \def\bbl@tempa{\bbl@nostdfont{}}%
4537   \bbl@foreach\bbl@font@fams{%       don't gather with prev for
4538     \bbl@ifunset{bbl@##1dflt@\languagename}%
4539       {\bbl@cs{famrst@##1}%
4540        \global\bbl@csarg\let{famrst@##1}\relax}%
4541       {\bbl@exp{% order is relevant. TODO: but sometimes wrong!
4542         \\\bbl@add\\\originalTeX{%
4543           \\\bbl@font@rst{\bbl@cl{##1dflt}}%
4544                        \<##1default>\<##1family>{##1}}%
4545         \\\bbl@font@set\<bbl@##1dflt@\languagename>% the main part!
4546                        \<##1default>\<##1family>}}}%
4547   \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.

```
4548 \ifx\f@family\@undefined\else   % if latex
4549   \ifcase\bbl@engine             % if pdftex
4550     \let\bbl@ckeckstdfonts\relax
4551   \else
4552     \def\bbl@ckeckstdfonts{%
4553       \begingroup
4554         \global\let\bbl@ckeckstdfonts\relax
4555         \let\bbl@tempa\@empty
4556         \bbl@foreach\bbl@font@fams{%
```

```
4557        \bbl@ifunset{bbl@##1dflt@}%
4558          {\@nameuse{##1family}%
4559           \bbl@csarg\gdef{WFF@\f@family}{}% Flag
4560           \bbl@exp{\\\bbl@add\\\bbl@tempa{* \<##1family>= \f@family\\\\%
4561              \space\space\fontname\font\\\\}}%
4562           \bbl@csarg\xdef{##1dflt@}{\f@family}%
4563           \expandafter\xdef\csname ##1default\endcsname{\f@family}}%
4564          {}}%
4565        \ifx\bbl@tempa\@empty\else
4566          \bbl@infowarn{The following font families will use the default\\%
4567            settings for all or some languages:\\%
4568            \bbl@tempa
4569            There is nothing intrinsically wrong with it, but\\%
4570            'babel' will no set Script and Language, which could\\%
4571             be relevant in some languages. If your document uses\\%
4572             these families, consider redefining them with \string\babelfont.\\%
4573            Reported}%
4574        \fi
4575      \endgroup}
4576   \fi
4577 \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.

```
4578 \def\bbl@font@set#1#2#3{% eg \bbl@rmdflt@lang \rmdefault \rmfamily
4579   \bbl@xin@{<>}{#1}%
4580   \ifin@
4581     \bbl@exp{\\\bbl@fontspec@set\\#1\expandafter\@gobbletwo#1\\#3}%
4582   \fi
4583   \bbl@exp{%                'Unprotected' macros return prev values
4584     \def\\#2{#1}%            eg, \rmdefault{\bbl@rmdflt@lang}
4585     \\\bbl@ifsamestring{#2}{\f@family}%
4586       {\\#3%
4587        \\\bbl@ifsamestring{\f@series}{\bfdefault}{\\\bfseries}{}%
4588        \let\\\bbl@tempa\relax}%
4589       {}}}
4590 %    TODO - next should be global?, but even local does its job. I'm
4591 %    still not sure -- must investigate:
4592 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
4593   \let\bbl@tempe\bbl@mapselect
4594   \let\bbl@mapselect\relax
4595   \let\bbl@temp@fam#4%        eg, '\rmfamily', to be restored below
4596   \let#4\@empty     %        Make sure \renewfontfamily is valid
4597   \bbl@exp{%
4598     \let\\\bbl@temp@pfam\<\bbl@stripslash#4\space>% eg, '\rmfamily '
4599     \<keys_if_exist:nnF>{fontspec-opentype}{Script/\bbl@cl{sname}}%
4600       {\\\newfontscript{\bbl@cl{sname}}{\bbl@cl{sotf}}}%
4601     \<keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@cl{lname}}%
4602       {\\\newfontlanguage{\bbl@cl{lname}}{\bbl@cl{lotf}}}%
4603     \\\renewfontfamily\\#4%
4604       [\bbl@cl{lsys},#2]}{#3}% ie \bbl@exp{..}{#3}
4605   \begingroup
4606     #4%
4607     \xdef#1{\f@family}%      eg, \bbl@rmdflt@lang{FreeSerif(0)}
4608   \endgroup
4609   \let#4\bbl@temp@fam
4610   \bbl@exp{\let\<\bbl@stripslash#4\space>}\bbl@temp@pfam
4611   \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.

156

```
4612 \def\bbl@font@rst#1#2#3#4{%
4613   \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.

```
4614 \def\bbl@font@fams{rm,sf,tt}
```

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

```
4615 \newcommand\babelFSstore[2][]{%
4616   \bbl@ifblank{#1}%
4617     {\bbl@csarg\def{sname@#2}{Latin}}%
4618     {\bbl@csarg\def{sname@#2}{#1}}%
4619   \bbl@provide@dirs{#2}%
4620   \bbl@csarg\ifnum{wdir@#2}>\z@
4621     \let\bbl@beforeforeign\leavevmode
4622     \EnableBabelHook{babel-bidi}%
4623   \fi
4624   \bbl@foreach{#2}{%
4625     \bbl@FSstore{##1}{rm}\rmdefault\bbl@save@rmdefault
4626     \bbl@FSstore{##1}{sf}\sfdefault\bbl@save@sfdefault
4627     \bbl@FSstore{##1}{tt}\ttdefault\bbl@save@ttdefault}}
4628 \def\bbl@FSstore#1#2#3#4{%
4629   \bbl@csarg\edef{#2default#1}{#3}%
4630   \expandafter\addto\csname extras#1\endcsname{%
4631     \let#4#3%
4632     \ifx#3\f@family
4633       \edef#3{\csname bbl@#2default#1\endcsname}%
4634       \fontfamily{#3}\selectfont
4635     \else
4636       \edef#3{\csname bbl@#2default#1\endcsname}%
4637     \fi}%
4638   \expandafter\addto\csname noextras#1\endcsname{%
4639     \ifx#3\f@family
4640       \fontfamily{#4}\selectfont
4641     \fi
4642     \let#3#4}}
4643 \let\bbl@langfeatures\@empty
4644 \def\babelFSfeatures{% make sure \fontspec is redefined once
4645   \let\bbl@ori@fontspec\fontspec
4646   \renewcommand\fontspec[1][]{%
4647     \bbl@ori@fontspec[\bbl@langfeatures##1]}
4648   \let\babelFSfeatures\bbl@FSfeatures
4649   \babelFSfeatures}
4650 \def\bbl@FSfeatures#1#2{%
4651   \expandafter\addto\csname extras#1\endcsname{%
4652     \babel@save\bbl@langfeatures
4653     \edef\bbl@langfeatures{#2,}}}
4654 ⟨⟨/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.

```
4655 ⟨⟨*Footnote changes⟩⟩ ≡
4656 \bbl@trace{Bidi footnotes}
4657 \ifnum\bbl@bidimode>\z@
4658   \def\bbl@footnote#1#2#3{%
4659     \@ifnextchar[%
4660       {\bbl@footnote@o{#1}{#2}{#3}}%
```

```
4661        {\bbl@footnote@x{#1}{#2}{#3}}}
4662    \long\def\bbl@footnote@x#1#2#3#4{%
4663      \bgroup
4664        \select@language@x{\bbl@main@language}%
4665        \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
4666      \egroup}
4667    \long\def\bbl@footnote@o#1#2#3[#4]#5{%
4668      \bgroup
4669        \select@language@x{\bbl@main@language}%
4670        \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
4671      \egroup}
4672    \def\bbl@footnotetext#1#2#3{%
4673      \@ifnextchar[%
4674        {\bbl@footnotetext@o{#1}{#2}{#3}}%
4675        {\bbl@footnotetext@x{#1}{#2}{#3}}}
4676    \long\def\bbl@footnotetext@x#1#2#3#4{%
4677      \bgroup
4678        \select@language@x{\bbl@main@language}%
4679        \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
4680      \egroup}
4681    \long\def\bbl@footnotetext@o#1#2#3[#4]#5{%
4682      \bgroup
4683        \select@language@x{\bbl@main@language}%
4684        \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
4685      \egroup}
4686    \def\BabelFootnote#1#2#3#4{%
4687      \ifx\bbl@fn@footnote\@undefined
4688        \let\bbl@fn@footnote\footnote
4689      \fi
4690      \ifx\bbl@fn@footnotetext\@undefined
4691        \let\bbl@fn@footnotetext\footnotetext
4692      \fi
4693      \bbl@ifblank{#2}%
4694        {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
4695         \@namedef{\bbl@stripslash#1text}%
4696           {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
4697        {\def#1{\bbl@exp{\\\bbl@footnote{\\\foreignlanguage{#2}}}{#3}{#4}}%
4698         \@namedef{\bbl@stripslash#1text}%
4699           {\bbl@exp{\\\bbl@footnotetext{\\\foreignlanguage{#2}}}{#3}{#4}}}}
4700 \fi
4701 ⟨⟨/Footnote changes⟩⟩
```

Now, the code.

```
4702 ⟨*xetex⟩
4703 \def\BabelStringsDefault{unicode}
4704 \let\xebbl@stop\relax
4705 \AddBabelHook{xetex}{encodedcommands}{%
4706   \def\bbl@tempa{#1}%
4707   \ifx\bbl@tempa\@empty
4708     \XeTeXinputencoding"bytes"%
4709   \else
4710     \XeTeXinputencoding"#1"%
4711   \fi
4712   \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4713 \AddBabelHook{xetex}{stopcommands}{%
4714   \xebbl@stop
4715   \let\xebbl@stop\relax}
4716 \def\bbl@intraspace#1 #2 #3\@@{%
4717   \bbl@csarg\gdef{xeisp@\languagename}%
4718     {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4719 \def\bbl@intrapenalty#1\@@{%
4720   \bbl@csarg\gdef{xeipn@\languagename}%
4721     {\XeTeXlinebreakpenalty #1\relax}}
```

```
4722 \def\bbl@provide@intraspace{%
4723   \bbl@xin@{/s}{/\bbl@cl{lnbrk}}%
4724   \ifin@\else\bbl@xin@{/c}{/\bbl@cl{lnbrk}}\fi
4725   \ifin@
4726     \bbl@ifunset{bbl@intsp@\languagename}{}%
4727       {\expandafter\ifx\csname bbl@intsp@\languagename\endcsname\@empty\else
4728         \ifx\bbl@KVP@intraspace\@nil
4729           \bbl@exp{%
4730             \\\bbl@intraspace\bbl@cl{intsp}\\\@@}%
4731         \fi
4732         \ifx\bbl@KVP@intrapenalty\@nil
4733           \bbl@intrapenalty0\@@
4734         \fi
4735       \fi
4736       \ifx\bbl@KVP@intraspace\@nil\else % We may override the ini
4737         \expandafter\bbl@intraspace\bbl@KVP@intraspace\@@
4738       \fi
4739       \ifx\bbl@KVP@intrapenalty\@nil\else
4740         \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
4741       \fi
4742       \bbl@exp{%
4743         % TODO. Execute only once (but redundant):
4744         \\\bbl@add\<extras\languagename>{%
4745           \XeTeXlinebreaklocale "\bbl@cl{tbcp}"%
4746           \<bbl@xeisp@\languagename>%
4747           \<bbl@xeipn@\languagename>}%
4748         \\\bbl@toglobal\<extras\languagename>%
4749         \\\bbl@add\<noextras\languagename>{%
4750           \XeTeXlinebreaklocale "en"}%
4751         \\\bbl@toglobal\<noextras\languagename>}%
4752       \ifx\bbl@ispacesize\@undefined
4753         \gdef\bbl@ispacesize{\bbl@cl{xeisp}}%
4754         \ifx\AtBeginDocument\@notprerr
4755           \expandafter\@secondoftwo  % to execute right now
4756         \fi
4757         \AtBeginDocument{\bbl@patchfont{\bbl@ispacesize}}%
4758       \fi}%
4759   \fi}
4760 \ifx\DisableBabelHook\@undefined\endinput\fi
4761 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4762 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
4763 \DisableBabelHook{babel-fontspec}
4764 ⟨⟨Font selection⟩⟩
4765 \input txtbabel.def
4766 ⟨/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.

```
4767 ⟨*texxet⟩
4768 \providecommand\bbl@provide@intraspace{}
4769 \bbl@trace{Redefinitions for bidi layout}
4770 \def\bbl@sspre@caption{%
4771   \bbl@exp{\everyhbox{\\\bbl@textdir\bbl@cs{wdir@\bbl@main@language}}}}
4772 \ifx\bbl@opt@layout\@nnil\endinput\fi  % No layout
4773 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
4774 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
```

```
4775 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
4776   \def\@hangfrom#1{%
4777     \setbox\@tempboxa\hbox{{#1}}%
4778     \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
4779     \noindent\box\@tempboxa}
4780   \def\raggedright{%
4781     \let\\\@centercr
4782     \bbl@startskip\z@skip
4783     \@rightskip\@flushglue
4784     \bbl@endskip\@rightskip
4785     \parindent\z@
4786     \parfillskip\bbl@startskip}
4787   \def\raggedleft{%
4788     \let\\\@centercr
4789     \bbl@startskip\@flushglue
4790     \bbl@endskip\z@skip
4791     \parindent\z@
4792     \parfillskip\bbl@endskip}
4793 \fi
4794 \IfBabelLayout{lists}
4795   {\bbl@sreplace\list
4796     {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
4797   \def\bbl@listleftmargin{%
4798     \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
4799   \ifcase\bbl@engine
4800     \def\labelenumii{)\theenumii(}% pdftex doesn't reverse ()
4801     \def\p@enumiii{\p@enumii)\theenumii(}%
4802   \fi
4803   \bbl@sreplace\@verbatim
4804     {\leftskip\@totalleftmargin}%
4805     {\bbl@startskip\textwidth
4806      \advance\bbl@startskip-\linewidth}%
4807   \bbl@sreplace\@verbatim
4808     {\rightskip\z@skip}%
4809     {\bbl@endskip\z@skip}}%
4810   {}
4811 \IfBabelLayout{contents}
4812   {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
4813   \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
4814   {}
4815 \IfBabelLayout{columns}
4816   {\bbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbl@outputhbox}%
4817   \def\bbl@outputhbox#1{%
4818     \hb@xt@\textwidth{%
4819       \hskip\columnwidth
4820       \hfil
4821       {\normalcolor\vrule \@width\columnseprule}%
4822       \hfil
4823       \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
4824       \hskip-\textwidth
4825       \hb@xt@\columnwidth{\box\@outputbox \hss}%
4826       \hskip\columnsep
4827       \hskip\columnwidth}}%
4828   {}
4829 ⟨⟨Footnote changes⟩⟩
4830 \IfBabelLayout{footnotes}%
4831   {\BabelFootnote\footnote\languagename{}{}%
4832   \BabelFootnote\localfootnote\languagename{}{}%
4833   \BabelFootnote\mainfootnote{}{}{}}
4834   {}
```

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.

```
4835 \IfBabelLayout{counters}%
4836   {\let\bbl@latinarabic=\@arabic
4837    \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
4838    \let\bbl@asciiroman=\@roman
4839    \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciiroman#1}}}%
4840    \let\bbl@asciiRoman=\@Roman
4841    \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}}{}
4842 ⟨/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`).

```
4843 ⟨*luatex⟩
4844 \ifx\AddBabelHook\@undefined % When plain.def, babel.sty starts
4845 \bbl@trace{Read language.dat}
4846 \ifx\bbl@readstream\@undefined
4847   \csname newread\endcsname\bbl@readstream
4848 \fi
4849 \begingroup
4850   \toks@{}
4851   \count@\z@ % 0=start, 1=0th, 2=normal
4852   \def\bbl@process@line#1#2 #3 #4 {%
4853     \ifx=#1%
4854       \bbl@process@synonym{#2}%
4855     \else
4856       \bbl@process@language{#1#2}{#3}{#4}%
4857     \fi
4858     \ignorespaces}
4859   \def\bbl@manylang{%
4860     \ifnum\bbl@last>\@ne
4861       \bbl@info{Non-standard hyphenation setup}%
4862     \fi
```

```
4863        \let\bbl@manylang\relax}
4864    \def\bbl@process@language#1#2#3{%
4865        \ifcase\count@
4866            \@ifundefined{zth@#1}{\count@\tw@}{\count@\@ne}%
4867        \or
4868            \count@\tw@
4869        \fi
4870        \ifnum\count@=\tw@
4871            \expandafter\addlanguage\csname l@#1\endcsname
4872            \language\allocationnumber
4873            \chardef\bbl@last\allocationnumber
4874            \bbl@manylang
4875            \let\bbl@elt\relax
4876            \xdef\bbl@languages{%
4877                \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
4878        \fi
4879        \the\toks@
4880        \toks@{}}
4881    \def\bbl@process@synonym@aux#1#2{%
4882        \global\expandafter\chardef\csname l@#1\endcsname#2\relax
4883        \let\bbl@elt\relax
4884        \xdef\bbl@languages{%
4885            \bbl@languages\bbl@elt{#1}{#2}{}{}}%
4886    \def\bbl@process@synonym#1{%
4887        \ifcase\count@
4888            \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
4889        \or
4890            \@ifundefined{zth@#1}{\bbl@process@synonym@aux{#1}{0}}{}%
4891        \else
4892            \bbl@process@synonym@aux{#1}{\the\bbl@last}%
4893        \fi}
4894    \ifx\bbl@languages\@undefined % Just a (sensible?) guess
4895        \chardef\l@english\z@
4896        \chardef\l@USenglish\z@
4897        \chardef\bbl@last\z@
4898        \global\@namedef{bbl@hyphendata@0}{{hyphen.tex}{}}
4899        \gdef\bbl@languages{%
4900            \bbl@elt{english}{0}{hyphen.tex}{}%
4901            \bbl@elt{USenglish}{0}{}{}}
4902    \else
4903        \global\let\bbl@languages@format\bbl@languages
4904        \def\bbl@elt#1#2#3#4{% Remove all except language 0
4905            \ifnum#2>\z@\else
4906                \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
4907            \fi}%
4908        \xdef\bbl@languages{\bbl@languages}%
4909    \fi
4910    \def\bbl@elt#1#2#3#4{\@namedef{zth@#1}{}} % Define flags
4911    \bbl@languages
4912    \openin\bbl@readstream=language.dat
4913    \ifeof\bbl@readstream
4914        \bbl@warning{I couldn't find language.dat. No additional\\%
4915                    patterns loaded. Reported}%
4916    \else
4917        \loop
4918            \endlinechar\m@ne
4919            \read\bbl@readstream to \bbl@line
4920            \endlinechar`\^^M
4921            \if T\ifeof\bbl@readstream F\fi T\relax
4922                \ifx\bbl@line\@empty\else
4923                    \edef\bbl@line{\bbl@line\space\space\space}%
4924                    \expandafter\bbl@process@line\bbl@line\relax
4925                \fi
```

```
4926    \repeat
4927  \fi
4928 \endgroup
4929 \bbl@trace{Macros for reading patterns files}
4930 \def\bbl@get@enc#1:#2:#3\@@@{\def\bbl@hyph@enc{#2}}
4931 \ifx\babelcatcodetablenum\@undefined
4932  \ifx\newcatcodetable\@undefined
4933    \def\babelcatcodetablenum{5211}
4934    \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4935  \else
4936    \newcatcodetable\babelcatcodetablenum
4937    \newcatcodetable\bbl@pattcodes
4938  \fi
4939 \else
4940  \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4941 \fi
4942 \def\bbl@luapatterns#1#2{%
4943  \bbl@get@enc#1::\@@@
4944  \setbox\z@\hbox\bgroup
4945    \begingroup
4946      \savecatcodetable\babelcatcodetablenum\relax
4947      \initcatcodetable\bbl@pattcodes\relax
4948      \catcodetable\bbl@pattcodes\relax
4949        \catcode`\#=6  \catcode`\$=3 \catcode`\&=4 \catcode`\^=7
4950        \catcode`\_=8  \catcode`\{=1 \catcode`\}=2 \catcode`\~=13
4951        \catcode`\@=11 \catcode`\^^I=10 \catcode`\^^J=12
4952        \catcode`\<=12 \catcode`\>=12 \catcode`\*=12 \catcode`\.=12
4953        \catcode`\-=12 \catcode`\/=12 \catcode`\[=12 \catcode`\]=12
4954        \catcode`\`=12 \catcode`\'=12 \catcode`\"=12
4955        \input #1\relax
4956      \catcodetable\babelcatcodetablenum\relax
4957    \endgroup
4958    \def\bbl@tempa{#2}%
4959    \ifx\bbl@tempa\@empty\else
4960      \input #2\relax
4961    \fi
4962  \egroup}%
4963 \def\bbl@patterns@lua#1{%
4964  \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
4965    \csname l@#1\endcsname
4966    \edef\bbl@tempa{#1}%
4967  \else
4968    \csname l@#1:\f@encoding\endcsname
4969    \edef\bbl@tempa{#1:\f@encoding}%
4970  \fi\relax
4971  \@namedef{lu@texhyphen@loaded@\the\language}{}% Temp
4972  \@ifundefined{bbl@hyphendata@\the\language}%
4973    {\def\bbl@elt##1##2##3##4{%
4974       \ifnum##2=\csname l@\bbl@tempa\endcsname % #2=spanish, dutch:OT1...
4975         \def\bbl@tempb{##3}%
4976         \ifx\bbl@tempb\@empty\else % if not a synonymous
4977           \def\bbl@tempc{{##3}{##4}}%
4978         \fi
4979         \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
4980       \fi}%
4981      \bbl@languages
4982      \@ifundefined{bbl@hyphendata@\the\language}%
4983        {\bbl@info{No hyphenation patterns were set for\\%
4984                   language '\bbl@tempa'. Reported}}%
4985        {\expandafter\expandafter\expandafter\bbl@luapatterns
4986           \csname bbl@hyphendata@\the\language\endcsname}}{}}
4987 \endinput\fi
4988  % Here ends \ifx\AddBabelHook\@undefined
```

```
4989   % A few lines are only read by hyphen.cfg
4990 \ifx\DisableBabelHook\@undefined
4991   \AddBabelHook{luatex}{everylanguage}{%
4992     \def\process@language##1##2##3{%
4993       \def\process@line####1####2 ####3 ####4 {}}}
4994   \AddBabelHook{luatex}{loadpatterns}{%
4995     \input #1\relax
4996     \expandafter\gdef\csname bbl@hyphendata@\the\language\endcsname
4997       {{#1}{}}}
4998   \AddBabelHook{luatex}{loadexceptions}{%
4999     \input #1\relax
5000     \def\bbl@tempb##1##2{{##1}{#1}}%
5001     \expandafter\xdef\csname bbl@hyphendata@\the\language\endcsname
5002       {\expandafter\expandafter\expandafter\bbl@tempb
5003        \csname bbl@hyphendata@\the\language\endcsname}}
5004 \endinput\fi
5005   % Here stops reading code for hyphen.cfg
5006   % The following is read the 2nd time it's loaded
5007 \begingroup  % TODO - to a lua file
5008 \catcode`\%=12
5009 \catcode`\'=12
5010 \catcode`\"=12
5011 \catcode`\:=12
5012 \directlua{
5013   Babel = Babel or {}
5014   function Babel.bytes(line)
5015     return line:gsub("(.)",
5016       function (chr) return unicode.utf8.char(string.byte(chr)) end)
5017   end
5018   function Babel.begin_process_input()
5019     if luatexbase and luatexbase.add_to_callback then
5020       luatexbase.add_to_callback('process_input_buffer',
5021                                  Babel.bytes,'Babel.bytes')
5022     else
5023       Babel.callback = callback.find('process_input_buffer')
5024       callback.register('process_input_buffer',Babel.bytes)
5025     end
5026   end
5027   function Babel.end_process_input ()
5028     if luatexbase and luatexbase.remove_from_callback then
5029       luatexbase.remove_from_callback('process_input_buffer','Babel.bytes')
5030     else
5031       callback.register('process_input_buffer',Babel.callback)
5032     end
5033   end
5034   function Babel.addpatterns(pp, lg)
5035     local lg = lang.new(lg)
5036     local pats = lang.patterns(lg) or ''
5037     lang.clear_patterns(lg)
5038     for p in pp:gmatch('[^%s]+') do
5039       ss = ''
5040       for i in string.utfcharacters(p:gsub('%d', '')) do
5041         ss = ss .. '%d?' .. i
5042       end
5043       ss = ss:gsub('^%%d%?%.', '%%.') .. '%d?'
5044       ss = ss:gsub('%.%%d%?$', '%%.')
5045       pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
5046       if n == 0 then
5047         tex.sprint(
5048           [[\string\csname\space bbl@info\endcsname{New pattern: ]]
5049           .. p .. [[}]])
5050         pats = pats .. ' ' .. p
5051       else
```

```
5052        tex.sprint(
5053          [[\string\csname\space bbl@info\endcsname{Renew pattern: ]]
5054          .. p .. [[}]])
5055      end
5056    end
5057    lang.patterns(lg, pats)
5058  end
5059  function Babel.hlist_has_bidi(head)
5060    local has_bidi = false
5061    for item in node.traverse(head) do
5062      if item.id == node.id'glyph' then
5063        local itemchar = item.char
5064        local chardata = Babel.characters[itemchar]
5065        local dir = chardata and chardata.d or nil
5066        if not dir then
5067          for nn, et in ipairs(Babel.ranges) do
5068            if itemchar < et[1] then
5069              break
5070            elseif itemchar <= et[2] then
5071              dir = et[3]
5072              break
5073            end
5074          end
5075        end
5076        if dir and (dir == 'al' or dir == 'r') then
5077          has_bidi = true
5078        end
5079      end
5080    end
5081    return has_bidi
5082  end
5083  function Babel.set_chranges_b (script, chrng)
5084    if chrng == '' then return end
5085    texio.write('Replacing ' .. script .. ' script ranges')
5086    Babel.script_blocks[script] = {}
5087    for s, e in string.gmatch(chrng..' ', '(.-)%.%.(.-)%s') do
5088      table.insert(
5089        Babel.script_blocks[script], {tonumber(s,16), tonumber(e,16)})
5090    end
5091  end
5092 }
5093 \endgroup
5094 \ifx\newattribute\@undefined\else
5095   \newattribute\bbl@attr@locale
5096   \directlua{ Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale' }
5097   \AddBabelHook{luatex}{beforeextras}{%
5098     \setattribute\bbl@attr@locale\localeid}
5099 \fi
5100 \def\BabelStringsDefault{unicode}
5101 \let\luabbl@stop\relax
5102 \AddBabelHook{luatex}{encodedcommands}{%
5103   \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
5104   \ifx\bbl@tempa\bbl@tempb\else
5105     \directlua{Babel.begin_process_input()}%
5106     \def\luabbl@stop{%
5107       \directlua{Babel.end_process_input()}}%
5108   \fi}%
5109 \AddBabelHook{luatex}{stopcommands}{%
5110   \luabbl@stop
5111   \let\luabbl@stop\relax}
5112 \AddBabelHook{luatex}{patterns}{%
5113   \@ifundefined{bbl@hyphendata@\the\language}%
5114     {\def\bbl@elt##1##2##3##4{%
```

```
5115        \ifnum##2=\csname l@#2\endcsname % #2=spanish, dutch:OT1...
5116          \def\bbl@tempb{##3}%
5117          \ifx\bbl@tempb\@empty\else % if not a synonymous
5118            \def\bbl@tempc{{##3}{##4}}%
5119          \fi
5120          \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5121        \fi}%
5122      \bbl@languages
5123      \@ifundefined{bbl@hyphendata@\the\language}%
5124        {\bbl@info{No hyphenation patterns were set for\\%
5125                  language '#2'. Reported}}%
5126        {\expandafter\expandafter\expandafter\bbl@luapatterns
5127          \csname bbl@hyphendata@\the\language\endcsname}}{}%
5128  \@ifundefined{bbl@patterns@}{}{%
5129    \begingroup
5130      \bbl@xin@{,\number\language,}{,\bbl@pttnlist}%
5131      \ifin@\else
5132        \ifx\bbl@patterns@\@empty\else
5133          \directlua{ Babel.addpatterns(
5134            [[\bbl@patterns@]], \number\language) }%
5135        \fi
5136        \@ifundefined{bbl@patterns@#1}%
5137          \@empty
5138          {\directlua{ Babel.addpatterns(
5139                [[\space\csname bbl@patterns@#1\endcsname]],
5140                \number\language) }}%
5141        \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
5142      \fi
5143    \endgroup}%
5144  \bbl@exp{%
5145    \bbl@ifunset{bbl@prehc@\languagename}{}%
5146      {\\\bbl@ifblank{\bbl@cs{prehc@\languagename}}{}%
5147        {\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.

```
5148  \@onlypreamble\babelpatterns
5149  \AtEndOfPackage{%
5150    \newcommand\babelpatterns[2][\@empty]{%
5151      \ifx\bbl@patterns@\relax
5152        \let\bbl@patterns@\@empty
5153      \fi
5154      \ifx\bbl@pttnlist\@empty\else
5155        \bbl@warning{%
5156          You must not intermingle \string\selectlanguage\space and\\%
5157          \string\babelpatterns\space or some patterns will not\\%
5158          be taken into account. Reported}%
5159      \fi
5160      \ifx\@empty#1%
5161        \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
5162      \else
5163        \edef\bbl@tempb{\zap@space#1 \@empty}%
5164        \bbl@for\bbl@tempa\bbl@tempb{%
5165          \bbl@fixname\bbl@tempa
5166          \bbl@iflanguage\bbl@tempa{%
5167            \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
5168              \@ifundefined{bbl@patterns@\bbl@tempa}%
5169                \@empty
5170                {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
5171              #2}}}%
5172      \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.

```
5173 % TODO - to a lua file
5174 \directlua{
5175   Babel = Babel or {}
5176   Babel.linebreaking = Babel.linebreaking or {}
5177   Babel.linebreaking.before = {}
5178   Babel.linebreaking.after = {}
5179   Babel.locale = {} % Free to use, indexed by \localeid
5180   function Babel.linebreaking.add_before(func)
5181     tex.print([[\noexpand\csname bbl@luahyphenate\endcsname]])
5182     table.insert(Babel.linebreaking.before, func)
5183   end
5184   function Babel.linebreaking.add_after(func)
5185     tex.print([[\noexpand\csname bbl@luahyphenate\endcsname]])
5186     table.insert(Babel.linebreaking.after, func)
5187   end
5188 }
5189 \def\bbl@intraspace#1 #2 #3\@@{%
5190   \directlua{
5191     Babel = Babel or {}
5192     Babel.intraspaces = Babel.intraspaces or {}
5193     Babel.intraspaces['\csname bbl@sbcp@\languagename\endcsname'] = %
5194       {b = #1, p = #2, m = #3}
5195     Babel.locale_props[\the\localeid].intraspace = %
5196       {b = #1, p = #2, m = #3}
5197   }}
5198 \def\bbl@intrapenalty#1\@@{%
5199   \directlua{
5200     Babel = Babel or {}
5201     Babel.intrapenalties = Babel.intrapenalties or {}
5202     Babel.intrapenalties['\csname bbl@sbcp@\languagename\endcsname'] = #1
5203     Babel.locale_props[\the\localeid].intrapenalty = #1
5204   }}
5205 \begingroup
5206 \catcode`\%=12
5207 \catcode`\^=14
5208 \catcode`\'=12
5209 \catcode`\~=12
5210 \gdef\bbl@seaintraspace{^
5211   \let\bbl@seaintraspace\relax
5212   \directlua{
5213     Babel = Babel or {}
5214     Babel.sea_enabled = true
5215     Babel.sea_ranges = Babel.sea_ranges or {}
5216     function Babel.set_chranges (script, chrng)
5217       local c = 0
5218       for s, e in string.gmatch(chrng..' ', '(.-)%.%.(.-)%s') do
5219         Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
5220         c = c + 1
5221       end
5222     end
5223     function Babel.sea_disc_to_space (head)
5224       local sea_ranges = Babel.sea_ranges
5225       local last_char = nil
5226       local quad = 655360      ^% 10 pt = 655360 = 10 * 65536
5227       for item in node.traverse(head) do
5228         local i = item.id
5229         if i == node.id'glyph' then
```

```
5230                 last_char = item
5231             elseif i == 7 and item.subtype == 3 and last_char
5232                 and last_char.char > 0x0C99 then
5233               quad = font.getfont(last_char.font).size
5234               for lg, rg in pairs(sea_ranges) do
5235                 if last_char.char > rg[1] and last_char.char < rg[2] then
5236                   lg = lg:sub(1, 4)  ^% Remove trailing number of, eg, Cyrl1
5237                   local intraspace = Babel.intraspaces[lg]
5238                   local intrapenalty = Babel.intrapenalties[lg]
5239                   local n
5240                   if intrapenalty ~= 0 then
5241                     n = node.new(14, 0)      ^% penalty
5242                     n.penalty = intrapenalty
5243                     node.insert_before(head, item, n)
5244                   end
5245                   n = node.new(12, 13)      ^% (glue, spaceskip)
5246                   node.setglue(n, intraspace.b * quad,
5247                                   intraspace.p * quad,
5248                                   intraspace.m * quad)
5249                   node.insert_before(head, item, n)
5250                   node.remove(head, item)
5251                 end
5252               end
5253             end
5254           end
5255       end
5256   }^^
5257   \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.

```
5258 \catcode`\%=14
5259 \gdef\bbl@cjkintraspace{%
5260   \let\bbl@cjkintraspace\relax
5261   \directlua{
5262     Babel = Babel or {}
5263     require('babel-data-cjk.lua')
5264     Babel.cjk_enabled = true
5265     function Babel.cjk_linebreak(head)
5266       local GLYPH = node.id'glyph'
5267       local last_char = nil
5268       local quad = 655360      % 10 pt = 655360 = 10 * 65536
5269       local last_class = nil
5270       local last_lang = nil
5271
5272       for item in node.traverse(head) do
5273         if item.id == GLYPH then
5274
5275           local lang = item.lang
5276
5277           local LOCALE = node.get_attribute(item,
5278               Babel.attr_locale)
5279           local props = Babel.locale_props[LOCALE]
5280
5281           local class = Babel.cjk_class[item.char].c
5282
5283           if props.cjk_quotes and props.cjk_quotes[item.char] then
```

```
5284                class = props.cjk_quotes[item.char]
5285              end
5286
5287            if class == 'cp' then class = 'cl' end % )] as CL
5288            if class == 'id' then class = 'I' end
5289
5290            local br = 0
5291            if class and last_class and Babel.cjk_breaks[last_class][class] then
5292              br = Babel.cjk_breaks[last_class][class]
5293            end
5294
5295            if br == 1 and props.linebreak == 'c' and
5296                lang ~= \the\l@nohyphenation\space and
5297                last_lang ~= \the\l@nohyphenation then
5298              local intrapenalty = props.intrapenalty
5299              if intrapenalty ~= 0 then
5300                local n = node.new(14, 0)      % penalty
5301                n.penalty = intrapenalty
5302                node.insert_before(head, item, n)
5303              end
5304              local intraspace = props.intraspace
5305              local n = node.new(12, 13)       % (glue, spaceskip)
5306              node.setglue(n, intraspace.b * quad,
5307                              intraspace.p * quad,
5308                              intraspace.m * quad)
5309              node.insert_before(head, item, n)
5310            end
5311
5312            if font.getfont(item.font) then
5313              quad = font.getfont(item.font).size
5314            end
5315            last_class = class
5316            last_lang = lang
5317          else % if penalty, glue or anything else
5318            last_class = nil
5319          end
5320        end
5321      lang.hyphenate(head)
5322    end
5323  }%
5324  \bbl@luahyphenate}
5325 \gdef\bbl@luahyphenate{%
5326  \let\bbl@luahyphenate\relax
5327  \directlua{
5328    luatexbase.add_to_callback('hyphenate',
5329    function (head, tail)
5330      if Babel.linebreaking.before then
5331        for k, func in ipairs(Babel.linebreaking.before)  do
5332          func(head)
5333        end
5334      end
5335      if Babel.cjk_enabled then
5336        Babel.cjk_linebreak(head)
5337      end
5338      lang.hyphenate(head)
5339      if Babel.linebreaking.after then
5340        for k, func in ipairs(Babel.linebreaking.after)  do
5341          func(head)
5342        end
5343      end
5344      if Babel.sea_enabled then
5345        Babel.sea_disc_to_space(head)
5346      end
```

```
5347       end,
5348       'Babel.hyphenate')
5349   }
5350 }
5351 \endgroup
5352 \def\bbl@provide@intraspace{%
5353   \bbl@ifunset{bbl@intsp@\languagename}{}%
5354     {\expandafter\ifx\csname bbl@intsp@\languagename\endcsname\@empty\else
5355       \bbl@xin@{/c}{/\bbl@cl{lnbrk}}%
5356       \ifin@            % cjk
5357         \bbl@cjkintraspace
5358         \directlua{
5359             Babel = Babel or {}
5360             Babel.locale_props = Babel.locale_props or {}
5361             Babel.locale_props[\the\localeid].linebreak = 'c'
5362         }%
5363         \bbl@exp{\\\bbl@intraspace\bbl@cl{intsp}\\\@@}%
5364         \ifx\bbl@KVP@intrapenalty\@nil
5365           \bbl@intrapenalty0\@@
5366         \fi
5367       \else              % sea
5368         \bbl@seaintraspace
5369         \bbl@exp{\\\bbl@intraspace\bbl@cl{intsp}\\\@@}%
5370         \directlua{
5371           Babel = Babel or {}
5372           Babel.sea_ranges = Babel.sea_ranges or {}
5373           Babel.set_chranges('\bbl@cl{sbcp}',
5374                               '\bbl@cl{chrng}')
5375         }%
5376         \ifx\bbl@KVP@intrapenalty\@nil
5377           \bbl@intrapenalty0\@@
5378         \fi
5379       \fi
5380     \fi
5381     \ifx\bbl@KVP@intrapenalty\@nil\else
5382       \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
5383     \fi}}
```

## 12.6  Arabic justification

```
5384 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5385 \def\bblar@chars{%
5386   0628,0629,062A,062B,062C,062D,062E,062F,0630,0631,0632,0633,%
5387   0634,0635,0636,0637,0638,0639,063A,063B,063C,063D,063E,063F,%
5388   0640,0641,0642,0643,0644,0645,0646,0647,0649}
5389 \def\bblar@elongated{%
5390   0626,0628,062A,062B,0633,0634,0635,0636,063B,%
5391   063C,063D,063E,063F,0641,0642,0643,0644,0646,%
5392   0649,064A}
5393 \begingroup
5394   \catcode`\_=11 \catcode`:=11
5395   \gdef\bblar@nofswarn{\gdef\msg_warning:nnx##1##2##3{}}
5396 \endgroup
5397 \gdef\bbl@arabicjust{%
5398   \let\bbl@arabicjust\relax
5399   \newattribute\bblar@kashida
5400   \directlua{ Babel.attr_kashida = luatexbase.registernumber'bblar@kashida' }%
5401   \bblar@kashida=\z@
5402   \bbl@patchfont{{\bbl@parsejalt}}%
5403   \directlua{
5404     Babel.arabic.elong_map   = Babel.arabic.elong_map or {}
5405     Babel.arabic.elong_map[\the\localeid]   = {}
5406     luatexbase.add_to_callback('post_linebreak_filter',
```

```
5407        Babel.arabic.justify, 'Babel.arabic.justify')
5408      luatexbase.add_to_callback('hpack_filter',
5409        Babel.arabic.justify_hbox, 'Babel.arabic.justify_hbox')
5410  }}%
5411 % Save both node lists to make replacement. TODO. Save also widths to
5412 % make computations
5413 \def\bblar@fetchjalt#1#2#3#4{%
5414   \bbl@exp{\\\bbl@foreach{#1}}{%
5415     \bbl@ifunset{bblar@JE@##1}%
5416       {\setbox\z@\hbox{^^^^200d\char"##1#2}}%
5417       {\setbox\z@\hbox{^^^^200d\char"\@nameuse{bblar@JE@##1}#2}}%
5418     \directlua{%
5419       local last = nil
5420       for item in node.traverse(tex.box[0].head) do
5421         if item.id == node.id'glyph' and item.char > 0x600 and
5422             not (item.char == 0x200D) then
5423           last = item
5424         end
5425       end
5426       Babel.arabic.#3['##1#4'] = last.char
5427     }}}
5428 % Brute force. No rules at all, yet. The ideal: look at jalt table. And
5429 % perhaps other tables (falt?, cswh?). What about kaf? And diacritic
5430 % positioning?
5431 \gdef\bbl@parsejalt{%
5432   \ifx\addfontfeature\@undefined\else
5433     \bbl@xin@{/e}{/\bbl@cl{lnbrk}}%
5434     \ifin@
5435       \directlua{%
5436         if Babel.arabic.elong_map[\the\localeid][\fontid\font] == nil then
5437           Babel.arabic.elong_map[\the\localeid][\fontid\font] = {}
5438           tex.print([[\string\csname\space bbl@parsejalti\endcsname]])
5439         end
5440       }%
5441     \fi
5442   \fi}
5443 \gdef\bbl@parsejalti{%
5444   \begingroup
5445     \let\bbl@parsejalt\relax    % To avoid infinite loop
5446     \edef\bbl@tempb{\fontid\font}%
5447     \bblar@nofswarn
5448     \bblar@fetchjalt\bblar@elongated{}{from}{}%
5449     \bblar@fetchjalt\bblar@chars{^^^^064a}{from}{a}% Alef maksura
5450     \bblar@fetchjalt\bblar@chars{^^^^0649}{from}{y}% Yeh
5451     \addfontfeature{RawFeature=+jalt}%
5452   % \@namedef{bblar@JE@0643}{06AA}% todo: catch medial kaf
5453     \bblar@fetchjalt\bblar@elongated{}{dest}{}%
5454     \bblar@fetchjalt\bblar@chars{^^^^064a}{dest}{a}%
5455     \bblar@fetchjalt\bblar@chars{^^^^0649}{dest}{y}%
5456     \directlua{%
5457       for k, v in pairs(Babel.arabic.from) do
5458         if Babel.arabic.dest[k] and
5459             not (Babel.arabic.from[k] == Babel.arabic.dest[k]) then
5460           Babel.arabic.elong_map[\the\localeid][\bbl@tempb]
5461             [Babel.arabic.from[k]] = Babel.arabic.dest[k]
5462         end
5463       end
5464     }%
5465   \endgroup}
5466 %
5467 \begingroup
5468 \catcode`#=11
5469 \catcode`~=11
```

```lua
\directlua{

Babel.arabic = Babel.arabic or {}
Babel.arabic.from = {}
Babel.arabic.dest = {}
Babel.arabic.justify_factor = 0.95
Babel.arabic.justify_enabled = true

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

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

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

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

  % Exclude last line. todo. But-- it discards one-word lines, too!
  % ? Look for glue = 12:15
  if (line.glue_sign == 1 and line.glue_order == 0) then
    elongs = {}      % Stores elongated candidates of each line
    k_list = {}      % And all letters with kashida
    pos_inline = 0   % Not yet used

    for n in node.traverse_id(GLYPH, line.head) do
      pos_inline = pos_inline + 1 % To find where it is. Not used.

      % Elongated glyphs
      if elong_map then
        local locale = node.get_attribute(n, LOCALE)
        if elong_map[locale] and elong_map[locale][n.font] and
```

172

```
5533          elong_map[locale][n.font][n.char] then
5534        table.insert(elongs, {node = n, locale = locale} )
5535        node.set_attribute(n.prev, KASHIDA, 0)
5536      end
5537    end
5538
5539    % Tatwil
5540    if Babel.kashida_wts then
5541      local k_wt = node.get_attribute(n, KASHIDA)
5542      if k_wt > 0 then % todo. parameter for multi inserts
5543        table.insert(k_list, {node = n, weight = k_wt, pos = pos_inline})
5544      end
5545    end
5546
5547  end % of node.traverse_id
5548
5549  if #elongs == 0 and #k_list == 0 then goto next_line end
5550  full  = line.width
5551  shift = line.shift
5552  goal  = full * Babel.arabic.justify_factor % A bit crude
5553  width = node.dimensions(line.head)    % The 'natural' width
5554
5555  % == Elongated ==
5556  % Original idea taken from 'chikenize'
5557  while (#elongs > 0 and width < goal) do
5558    subst_done = true
5559    local x = #elongs
5560    local curr = elongs[x].node
5561    local oldchar = curr.char
5562    curr.char = elong_map[elongs[x].locale][curr.font][curr.char]
5563    width = node.dimensions(line.head)  % Check if the line is too wide
5564    % Substitute back if the line would be too wide and break:
5565    if width > goal then
5566      curr.char = oldchar
5567      break
5568    end
5569    % If continue, pop the just substituted node from the list:
5570    table.remove(elongs, x)
5571  end
5572
5573  % == Tatwil ==
5574  if #k_list == 0 then goto next_line end
5575
5576  width = node.dimensions(line.head)    % The 'natural' width
5577  k_curr = #k_list
5578  wt_pos = 1
5579
5580  while width < goal do
5581    subst_done = true
5582    k_item = k_list[k_curr].node
5583    if k_list[k_curr].weight == Babel.kashida_wts[wt_pos] then
5584      d = node.copy(k_item)
5585      d.char = 0x0640
5586      line.head, new = node.insert_after(line.head, k_item, d)
5587      width_new = node.dimensions(line.head)
5588      if width > goal or width == width_new then
5589        node.remove(line.head, new) % Better compute before
5590        break
5591      end
5592      width = width_new
5593    end
5594    if k_curr == 1 then
5595      k_curr = #k_list
```

```
5596        wt_pos = (wt_pos >= table.getn(Babel.kashida_wts)) and 1 or wt_pos+1
5597      else
5598        k_curr = k_curr - 1
5599      end
5600    end
5601
5602    ::next_line::
5603
5604    % Must take into account marks and ins, see luatex manual.
5605    % Have to be executed only if there are changes. Investigate
5606    % what's going on exactly.
5607    if subst_done and not gc then
5608      d = node.hpack(line.head, full, 'exactly')
5609      d.shift = shift
5610      node.insert_before(head, line, d)
5611      node.remove(head, line)
5612    end
5613  end % if process line
5614 end
5615 }
5616 \endgroup
5617 \fi\fi % Arabic just block
```

## 12.7   Common stuff

```
5618 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
5619 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
5620 \DisableBabelHook{babel-fontspec}
5621 ⟨⟨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.

```
5622 % TODO - to a lua file
5623 \directlua{
5624 Babel.script_blocks = {
5625   ['dflt'] = {},
5626   ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
5627               {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
5628   ['Armn'] = {{0x0530, 0x058F}},
5629   ['Beng'] = {{0x0980, 0x09FF}},
5630   ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
5631   ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
5632   ['Cyrl'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
5633               {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
5634   ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
5635   ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
5636               {0xAB00, 0xAB2F}},
5637   ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
5638   % Don't follow strictly Unicode, which places some Coptic letters in
5639   % the 'Greek and Coptic' block
5640   ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
5641   ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
5642               {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
5643               {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
5644               {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
5645               {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
5646               {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
```

```
5647   ['Hebr'] = {{0x0590, 0x05FF}},
5648   ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
5649              {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
5650   ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
5651   ['Knda'] = {{0x0C80, 0x0CFF}},
5652   ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
5653              {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
5654              {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
5655   ['Laoo'] = {{0x0E80, 0x0EFF}},
5656   ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
5657              {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
5658              {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
5659   ['Mahj'] = {{0x11150, 0x1117F}},
5660   ['Mlym'] = {{0x0D00, 0x0D7F}},
5661   ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
5662   ['Orya'] = {{0x0B00, 0x0B7F}},
5663   ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
5664   ['Syrc'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
5665   ['Taml'] = {{0x0B80, 0x0BFF}},
5666   ['Telu'] = {{0x0C00, 0x0C7F}},
5667   ['Tfng'] = {{0x2D30, 0x2D7F}},
5668   ['Thai'] = {{0x0E00, 0x0E7F}},
5669   ['Tibt'] = {{0x0F00, 0x0FFF}},
5670   ['Vaii'] = {{0xA500, 0xA63F}},
5671   ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
5672 }
5673
5674 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
5675 Babel.script_blocks.Hant = Babel.script_blocks.Hans
5676 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
5677
5678 function Babel.locale_map(head)
5679   if not Babel.locale_mapped then return head end
5680
5681   local LOCALE = Babel.attr_locale
5682   local GLYPH = node.id('glyph')
5683   local inmath = false
5684   local toloc_save
5685   for item in node.traverse(head) do
5686     local toloc
5687     if not inmath and item.id == GLYPH then
5688       % Optimization: build a table with the chars found
5689       if Babel.chr_to_loc[item.char] then
5690         toloc = Babel.chr_to_loc[item.char]
5691       else
5692         for lc, maps in pairs(Babel.loc_to_scr) do
5693           for _, rg in pairs(maps) do
5694             if item.char >= rg[1] and item.char <= rg[2] then
5695               Babel.chr_to_loc[item.char] = lc
5696               toloc = lc
5697               break
5698             end
5699           end
5700         end
5701       end
5702       % Now, take action, but treat composite chars in a different
5703       % fashion, because they 'inherit' the previous locale. Not yet
5704       % optimized.
5705       if not toloc and
5706           (item.char >= 0x0300 and item.char <= 0x036F) or
5707           (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
5708           (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
5709         toloc = toloc_save
```

```
5710        end
5711      if toloc and toloc > -1 then
5712        if Babel.locale_props[toloc].lg then
5713          item.lang = Babel.locale_props[toloc].lg
5714          node.set_attribute(item, LOCALE, toloc)
5715        end
5716        if Babel.locale_props[toloc]['/'..item.font] then
5717          item.font = Babel.locale_props[toloc]['/'..item.font]
5718        end
5719        toloc_save = toloc
5720      end
5721    elseif not inmath and item.id == 7 then
5722      item.replace = item.replace and Babel.locale_map(item.replace)
5723      item.pre     = item.pre and Babel.locale_map(item.pre)
5724      item.post    = item.post and Babel.locale_map(item.post)
5725    elseif item.id == node.id'math' then
5726      inmath = (item.subtype == 0)
5727    end
5728  end
5729  return head
5730 end
5731 }
```

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

```
5732 \newcommand\babelcharproperty[1]{%
5733   \count@=#1\relax
5734   \ifvmode
5735     \expandafter\bbl@chprop
5736   \else
5737     \bbl@error{\string\babelcharproperty\space can be used only in\\%
5738                vertical mode (preamble or between paragraphs)}%
5739               {See the manual for futher info}%
5740   \fi}
5741 \newcommand\bbl@chprop[3][\the\count@]{%
5742   \@tempcnta=#1\relax
5743   \bbl@ifunset{bbl@chprop@#2}%
5744     {\bbl@error{No property named '#2'. Allowed values are\\%
5745                direction (bc), mirror (bmg), and linebreak (lb)}%
5746               {See the manual for futher info}}%
5747     {}%
5748   \loop
5749     \bbl@cs{chprop@#2}{#3}%
5750   \ifnum\count@<\@tempcnta
5751     \advance\count@\@ne
5752   \repeat}
5753 \def\bbl@chprop@direction#1{%
5754   \directlua{
5755     Babel.characters[\the\count@] =  Babel.characters[\the\count@] or {}
5756     Babel.characters[\the\count@]['d'] = '#1'
5757   }}
5758 \let\bbl@chprop@bc\bbl@chprop@direction
5759 \def\bbl@chprop@mirror#1{%
5760   \directlua{
5761     Babel.characters[\the\count@] =  Babel.characters[\the\count@] or {}
5762     Babel.characters[\the\count@]['m'] = '\number#1'
5763   }}
5764 \let\bbl@chprop@bmg\bbl@chprop@mirror
5765 \def\bbl@chprop@linebreak#1{%
5766   \directlua{
5767     Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
5768     Babel.cjk_characters[\the\count@]['c'] = '#1'
5769   }}
```

```
5770 \let\bbl@chprop@lb\bbl@chprop@linebreak
5771 \def\bbl@chprop@locale#1{%
5772   \directlua{
5773     Babel.chr_to_loc = Babel.chr_to_loc or {}
5774     Babel.chr_to_loc[\the\count@] =
5775       \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@@#1}}\space
5776   }}
```

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.

```
5777 \directlua{
5778   Babel.nohyphenation = \the\l@nohyphenation
5779 }
```

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).

```
5780 \begingroup
5781 \catcode`\~=12
5782 \catcode`\%=12
5783 \catcode`\&=14
5784 \catcode`\|=12
5785 \gdef\babelprehyphenation{&%
5786   \@ifnextchar[{\bbl@settransform{0}}{\bbl@settransform{0}[]}}
5787 \gdef\babelposthyphenation{&%
5788   \@ifnextchar[{\bbl@settransform{1}}{\bbl@settransform{1}[]}}
5789 \gdef\bbl@settransform#1[#2]#3#4#5{&%
5790   \ifcase#1
5791     \bbl@activateprehyphen
5792   \else
5793     \bbl@activateposthyphen
5794   \fi
5795   \begingroup
5796     \def\babeltempa{\bbl@add@list\babeltempb}&%
5797     \let\babeltempb\@empty
5798     \def\bbl@tempa{#5}&%
5799     \bbl@replace\bbl@tempa{,}{ ,}&% TODO. Ugly trick to preserve {}
5800     \expandafter\bbl@foreach\expandafter{\bbl@tempa}{&%
5801       \bbl@ifsamestring{##1}{remove}&%
5802         {\bbl@add@list\babeltempb{nil}}&%
5803         {\directlua{
5804            local rep = [=[##1]=]
5805            rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
5806            rep = rep:gsub('^%s*(insert)%s*,', 'insert = true, ')
5807            rep = rep:gsub('(string)%s*=%s*([^%s,]*)', Babel.capture_func)
5808            if #1 == 0 then
5809              rep = rep:gsub('(space)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
5810                'space = {' .. '%2, %3, %4' .. '}')
5811              rep = rep:gsub('(spacefactor)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
5812                'spacefactor = {' .. '%2, %3, %4' .. '}')
5813              rep = rep:gsub('(kashida)%s*=%s*([^%s,]*)', Babel.capture_kashida)
5814            else
5815              rep = rep:gsub(   '(no)%s*=%s*([^%s,]*)', Babel.capture_func)
5816              rep = rep:gsub(   '(pre)%s*=%s*([^%s,]*)', Babel.capture_func)
5817              rep = rep:gsub(   '(post)%s*=%s*([^%s,]*)', Babel.capture_func)
5818            end
5819            tex.print([[\string\babeltempa{{]] .. rep .. [[}}]])
5820         }}}&%
```

```
5821        \let\bbl@kv@attribute\relax
5822        \let\bbl@kv@label\relax
5823        \bbl@forkv{#2}{\bbl@csarg\edef{kv@##1}{##2}}&%
5824        \ifx\bbl@kv@attribute\relax\else
5825          \edef\bbl@kv@attribute{\expandafter\bbl@stripslash\bbl@kv@attribute}&%
5826        \fi
5827        \directlua{
5828          local lbkr = Babel.linebreaking.replacements[#1]
5829          local u = unicode.utf8
5830          local id, attr, label
5831          if #1 == 0 then
5832            id = \the\csname bbl@id@@#3\endcsname\space
5833          else
5834            id = \the\csname l@#3\endcsname\space
5835          end
5836          \ifx\bbl@kv@attribute\relax
5837            attr = -1
5838          \else
5839            attr = luatexbase.registernumber'\bbl@kv@attribute'
5840          \fi
5841          \ifx\bbl@kv@label\relax\else  &% Same refs:
5842            label = [==[\bbl@kv@label]==]
5843          \fi
5844          &% Convert pattern:
5845          local patt = string.gsub([==[#4]==], '%s', '')
5846          if #1 == 0 then
5847            patt = string.gsub(patt, '|', ' ')
5848          end
5849          if not u.find(patt, '()', nil, true) then
5850            patt = '()' .. patt .. '()'
5851          end
5852          if #1 == 1 then
5853            patt = string.gsub(patt, '%(%)%^', '^()')
5854            patt = string.gsub(patt, '%$%(%)', '()$')
5855          end
5856          patt = u.gsub(patt, '{(.)}',
5857                  function (n)
5858                    return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
5859                  end)
5860          patt = u.gsub(patt, '{(%x%x%x%x+)}',
5861                  function (n)
5862                    return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%1')
5863                  end)
5864          lbkr[id] = lbkr[id] or {}
5865          table.insert(lbkr[id],
5866            { label=label, attr=attr, pattern=patt, replace={\babeltempb} })
5867        }&%
5868      \endgroup}
5869  \endgroup
5870  \def\bbl@activateposthyphen{%
5871      \let\bbl@activateposthyphen\relax
5872      \directlua{
5873        require('babel-transforms.lua')
5874        Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
5875      }}
5876  \def\bbl@activateprehyphen{%
5877      \let\bbl@activateprehyphen\relax
5878      \directlua{
5879        require('babel-transforms.lua')
5880        Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
5881      }}
```

## 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.

```
5882 \def\bbl@activate@preotf{%
5883   \let\bbl@activate@preotf\relax  % only once
5884   \directlua{
5885     Babel = Babel or {}
5886     %
5887     function Babel.pre_otfload_v(head)
5888       if Babel.numbers and Babel.digits_mapped then
5889         head = Babel.numbers(head)
5890       end
5891       if Babel.bidi_enabled then
5892         head = Babel.bidi(head, false, dir)
5893       end
5894       return head
5895     end
5896     %
5897     function Babel.pre_otfload_h(head, gc, sz, pt, dir)
5898       if Babel.numbers and Babel.digits_mapped then
5899         head = Babel.numbers(head)
5900       end
5901       if Babel.bidi_enabled then
5902         head = Babel.bidi(head, false, dir)
5903       end
5904       return head
5905     end
5906     %
5907     luatexbase.add_to_callback('pre_linebreak_filter',
5908       Babel.pre_otfload_v,
5909       'Babel.pre_otfload_v',
5910       luatexbase.priority_in_callback('pre_linebreak_filter',
5911         'luaotfload.node_processor') or nil)
5912     %
5913     luatexbase.add_to_callback('hpack_filter',
5914       Babel.pre_otfload_h,
5915       'Babel.pre_otfload_h',
5916       luatexbase.priority_in_callback('hpack_filter',
5917         'luaotfload.node_processor') or nil)
5918   }}
```

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=.

```
5919 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5920   \let\bbl@beforeforeign\leavevmode
5921   \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
5922   \RequirePackage{luatexbase}
5923   \bbl@activate@preotf
5924   \directlua{
5925     require('babel-data-bidi.lua')
5926     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
5927       require('babel-bidi-basic.lua')
5928     \or
5929       require('babel-bidi-basic-r.lua')
5930     \fi}
5931   % TODO - to locale_props, not as separate attribute
5932   \newattribute\bbl@attr@dir
5933   \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
5934   % TODO. I don't like it, hackish:
5935   \bbl@exp{\output{\bodydir\pagedir\the\output}}
```

```
5936    \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
5937 \fi\fi
5938 \chardef\bbl@thetextdir\z@
5939 \chardef\bbl@thepardir\z@
5940 \def\bbl@getluadir#1{%
5941   \directlua{
5942     if tex.#1dir == 'TLT' then
5943       tex.sprint('0')
5944     elseif tex.#1dir == 'TRT' then
5945       tex.sprint('1')
5946     end}}
5947 \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
5948   \ifcase#3\relax
5949     \ifcase\bbl@getluadir{#1}\relax\else
5950       #2 TLT\relax
5951     \fi
5952   \else
5953     \ifcase\bbl@getluadir{#1}\relax
5954       #2 TRT\relax
5955     \fi
5956   \fi}
5957 \def\bbl@thedir{0}
5958 \def\bbl@textdir#1{%
5959   \bbl@setluadir{text}\textdir{#1}%
5960   \chardef\bbl@thetextdir#1\relax
5961   \edef\bbl@thedir{\the\numexpr\bbl@thepardir*3+#1}%
5962   \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*3+#1}}
5963 \def\bbl@pardir#1{%
5964   \bbl@setluadir{par}\pardir{#1}%
5965   \chardef\bbl@thepardir#1\relax}
5966 \def\bbl@bodydir{\bbl@setluadir{body}\bodydir}
5967 \def\bbl@pagedir{\bbl@setluadir{page}\pagedir}
5968 \def\bbl@dirparastext{\pardir\the\textdir\relax}%    %%%%
5969 %
5970 \ifnum\bbl@bidimode>\z@
5971   \def\bbl@insidemath{0}%
5972   \def\bbl@everymath{\def\bbl@insidemath{1}}
5973   \def\bbl@everydisplay{\def\bbl@insidemath{2}}
5974   \frozen@everymath\expandafter{%
5975     \expandafter\bbl@everymath\the\frozen@everymath}
5976   \frozen@everydisplay\expandafter{%
5977     \expandafter\bbl@everydisplay\the\frozen@everydisplay}
5978   \AtBeginDocument{
5979     \directlua{
5980       function Babel.math_box_dir(head)
5981         if not (token.get_macro('bbl@insidemath') == '0') then
5982           if Babel.hlist_has_bidi(head) then
5983             local d = node.new(node.id'dir')
5984             d.dir = '+TRT'
5985             node.insert_before(head, node.has_glyph(head), d)
5986             for item in node.traverse(head) do
5987               node.set_attribute(item,
5988                 Babel.attr_dir, token.get_macro('bbl@thedir'))
5989             end
5990           end
5991         end
5992         return head
5993       end
5994       luatexbase.add_to_callback("hpack_filter", Babel.math_box_dir,
5995         "Babel.math_box_dir", 0)
5996   }}%
5997 \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.

```
5998 \bbl@trace{Redefinitions for bidi layout}
5999 %
6000 ⟨⟨*More package options⟩⟩ ≡
6001 \chardef\bbl@eqnpos\z@
6002 \DeclareOption{leqno}{\chardef\bbl@eqnpos\@ne}
6003 \DeclareOption{fleqn}{\chardef\bbl@eqnpos\tw@}
6004 ⟨⟨/More package options⟩⟩
6005 %
6006 \def\BabelNoAMSMath{\let\bbl@noamsmath\relax}
6007 \ifnum\bbl@bidimode>\z@
6008   \ifx\matheqdirmode\@undefined\else
6009     \matheqdirmode\@ne
6010   \fi
6011   \let\bbl@eqnodir\relax
6012   \def\bbl@eqdel{()}
6013   \def\bbl@eqnum{%
6014     {\normalfont\normalcolor
6015       \expandafter\@firstoftwo\bbl@eqdel
6016       \theequation
6017       \expandafter\@secondoftwo\bbl@eqdel}}
6018   \def\bbl@puteqno#1{\eqno\hbox{#1}}
6019   \def\bbl@putleqno#1{\leqno\hbox{#1}}
6020   \def\bbl@eqno@flip#1{%
6021     \ifdim\predisplaysize=-\maxdimen
6022       \eqno
6023       \hb@xt@.01pt{\hb@xt@\displaywidth{\hss{#1}}\hss}%
6024     \else
6025       \leqno\hbox{#1}%
6026     \fi}
6027   \def\bbl@leqno@flip#1{%
6028     \ifdim\predisplaysize=-\maxdimen
6029       \leqno
6030       \hb@xt@.01pt{\hss\hb@xt@\displaywidth{{#1}\hss}}%
6031     \else
6032       \eqno\hbox{#1}%
6033     \fi}
6034   \AtBeginDocument{%
6035     \ifx\maketag@@@\@undefined % Normal equation, eqnarray
6036       \AddToHook{env/equation/begin}{%
6037         \ifnum\bbl@thetextdir>\z@
6038           \let\@eqnnum\bbl@eqnum
6039           \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6040           \chardef\bbl@thetextdir\z@
6041           \bbl@add\normalfont{\bbl@eqnodir}%
6042           \ifcase\bbl@eqnpos
6043             \let\bbl@puteqno\bbl@eqno@flip
6044           \or
6045             \let\bbl@puteqno\bbl@leqno@flip
6046           \fi
```

```
6047            \fi}%
6048         \ifnum\bbl@eqnpos=\tw@\else
6049           \def\endequation{\bbl@puteqno{\@eqnnum}$$\@ignoretrue}%
6050         \fi
6051         \AddToHook{env/eqnarray/begin}{%
6052           \ifnum\bbl@thetextdir>\z@
6053             \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6054             \chardef\bbl@thetextdir\z@
6055             \bbl@add\normalfont{\bbl@eqnodir}%
6056             \ifnum\bbl@eqnpos=\@ne
6057               \def\@eqnnum{%
6058                 \setbox\z@\hbox{\bbl@eqnum}%
6059                 \hbox to0.01pt{\hss\hbox to\displaywidth{\box\z@\hss}}}%
6060             \else
6061                 \let\@eqnnum\bbl@eqnum
6062             \fi
6063           \fi}
6064       % Hack. YA luatex bug?:
6065       \expandafter\bbl@sreplace\csname] \endcsname{$$}{\eqno\kern.001pt$$}%
6066     \else % amstex
6067       \ifx\bbl@noamsmath\@undefined
6068         \ifnum\bbl@eqnpos=\@ne
6069           \let\bbl@ams@lap\hbox
6070         \else
6071           \let\bbl@ams@lap\llap
6072         \fi
6073         \ExplSyntaxOn
6074         \bbl@sreplace\intertext@{\normalbaselines}%
6075           {\normalbaselines
6076            \ifx\bbl@eqnodir\relax\else\bbl@pardir\@ne\bbl@eqnodir\fi}%
6077         \ExplSyntaxOff
6078         \def\bbl@ams@tagbox#1#2{#1{\bbl@eqnodir#2}}% #1=hbox|@lap|flip
6079         \ifx\bbl@ams@lap\hbox % leqno
6080           \def\bbl@ams@flip#1{%
6081             \hbox to 0.01pt{\hss\hbox to\displaywidth{{#1}\hss}}}%
6082         \else % eqno
6083           \def\bbl@ams@flip#1{%
6084             \hbox to 0.01pt{\hbox to\displaywidth{\hss{#1}}\hss}}%
6085         \fi
6086         \def\bbl@ams@preset#1{%
6087           \ifnum\bbl@thetextdir>\z@
6088             \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6089             \bbl@sreplace\textdef@{\hbox}{\bbl@ams@tagbox\hbox}%
6090             \bbl@sreplace\maketag@@@{\hbox}{\bbl@ams@tagbox#1}%
6091           \fi}%
6092         \ifnum\bbl@eqnpos=\tw@\else
6093           \def\bbl@ams@equation{%
6094             \ifnum\bbl@thetextdir>\z@
6095               \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6096               \chardef\bbl@thetextdir\z@
6097               \bbl@add\normalfont{\bbl@eqnodir}%
6098               \ifcase\bbl@eqnpos
6099                 \def\veqno##1##2{\bbl@eqno@flip{##1##2}}%
6100               \or
6101                 \def\veqno##1##2{\bbl@leqno@flip{##1##2}}%
6102               \fi
6103             \fi}%
6104           \AddToHook{env/equation/begin}{\bbl@ams@equation}%
6105           \AddToHook{env/equation*/begin}{\bbl@ams@equation}%
6106         \fi
6107         \AddToHook{env/cases/begin}{\bbl@ams@preset\bbl@ams@lap}%
6108         \AddToHook{env/multline/begin}{\bbl@ams@preset\hbox}%
6109         \AddToHook{env/gather/begin}{\bbl@ams@preset\bbl@ams@lap}%
```

182

```
6110        \AddToHook{env/gather*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6111        \AddToHook{env/align/begin}{\bbl@ams@preset\bbl@ams@lap}%
6112        \AddToHook{env/align*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6113        \AddToHook{env/eqnalign/begin}{\bbl@ams@preset\hbox}%
6114        % Hackish, for proper alignment. Don't ask me why it works!:
6115        \bbl@exp{% Avoid a 'visible' conditional
6116          \\\AddToHook{env/align*/end}{\<iftag@>\<else>\\\tag*{}\<fi>}}%
6117        \AddToHook{env/flalign/begin}{\bbl@ams@preset\hbox}%
6118        \AddToHook{env/split/before}{%
6119          \ifnum\bbl@thetextdir>\z@
6120            \bbl@ifsamestring\@currenvir{equation}%
6121              {\ifx\bbl@ams@lap\hbox % leqno
6122                \def\bbl@ams@flip#1{%
6123                  \hbox to 0.01pt{\hbox to\displaywidth{{#1}\hss}\hss}}%
6124              \else
6125                \def\bbl@ams@flip#1{%
6126                  \hbox to 0.01pt{\hss\hbox to\displaywidth{\hss{#1}}}}%
6127              \fi}%
6128            {}%
6129          \fi}%
6130      \fi
6131    \fi}
6132 \fi
6133 \ifx\bbl@opt@layout\@nnil\endinput\fi  % if no layout
6134 \ifnum\bbl@bidimode>\z@
6135   \def\bbl@nextfake#1{%  non-local changes, use always inside a group!
6136     \bbl@exp{%
6137       \def\\\bbl@insidemath{0}%
6138       \mathdir\the\bodydir
6139       #1%                Once entered in math, set boxes to restore values
6140       \<ifmmode>%
6141         \everyvbox{%
6142           \the\everyvbox
6143           \bodydir\the\bodydir
6144           \mathdir\the\mathdir
6145           \everyhbox{\the\everyhbox}%
6146           \everyvbox{\the\everyvbox}}%
6147         \everyhbox{%
6148           \the\everyhbox
6149           \bodydir\the\bodydir
6150           \mathdir\the\mathdir
6151           \everyhbox{\the\everyhbox}%
6152           \everyvbox{\the\everyvbox}}%
6153       \<fi>}}%
6154   \def\@hangfrom#1{%
6155     \setbox\@tempboxa\hbox{{#1}}%
6156     \hangindent\wd\@tempboxa
6157     \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
6158       \shapemode\@ne
6159     \fi
6160     \noindent\box\@tempboxa}
6161 \fi
6162 \IfBabelLayout{tabular}
6163   {\let\bbl@OL@@tabular\@tabular
6164    \bbl@replace\@tabular{$}{\bbl@nextfake$}%
6165    \let\bbl@NL@@tabular\@tabular
6166    \AtBeginDocument{%
6167      \ifx\bbl@NL@@tabular\@tabular\else
6168        \bbl@replace\@tabular{$}{\bbl@nextfake$}%
6169        \let\bbl@NL@@tabular\@tabular
6170      \fi}}
6171   {}
6172 \IfBabelLayout{lists}
```

```
6173  {\let\bbl@OL@list\list
6174   \bbl@sreplace\list{\parshape}{\bbl@listparshape}%
6175   \let\bbl@NL@list\list
6176   \def\bbl@listparshape#1#2#3{%
6177     \parshape #1 #2 #3 %
6178     \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
6179       \shapemode\tw@
6180     \fi}}
6181   {}
6182 \IfBabelLayout{graphics}
6183   {\let\bbl@pictresetdir\relax
6184    \def\bbl@pictsetdir#1{%
6185      \ifcase\bbl@thetextdir
6186        \let\bbl@pictresetdir\relax
6187      \else
6188        \ifcase#1\bodydir TLT  % Remember this sets the inner boxes
6189          \or\textdir TLT
6190          \else\bodydir TLT \textdir TLT
6191        \fi
6192        % \(text|par)dir required in pgf:
6193        \def\bbl@pictresetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}%
6194      \fi}%
6195    \AddToHook{env/picture/begin}{\bbl@pictsetdir\tw@}%
6196    \directlua{
6197      Babel.get_picture_dir = true
6198      Babel.picture_has_bidi = 0
6199      %
6200      function Babel.picture_dir (head)
6201        if not Babel.get_picture_dir then return head end
6202        if Babel.hlist_has_bidi(head) then
6203          Babel.picture_has_bidi = 1
6204        end
6205        return head
6206      end
6207      luatexbase.add_to_callback("hpack_filter", Babel.picture_dir,
6208        "Babel.picture_dir")
6209    }%
6210    \AtBeginDocument{%
6211      \long\def\put(#1,#2)#3{%
6212        \@killglue
6213        % Try:
6214        \ifx\bbl@pictresetdir\relax
6215          \def\bbl@tempc{0}%
6216        \else
6217          \directlua{
6218            Babel.get_picture_dir = true
6219            Babel.picture_has_bidi = 0
6220          }%
6221          \setbox\z@\hb@xt@\z@{%
6222            \@defaultunitsset\@tempdimc{#1}\unitlength
6223            \kern\@tempdimc
6224            #3\hss}% TODO: #3 executed twice (below). That's bad.
6225          \edef\bbl@tempc{\directlua{tex.print(Babel.picture_has_bidi)}}%
6226        \fi
6227        % Do:
6228        \@defaultunitsset\@tempdimc{#2}\unitlength
6229        \raise\@tempdimc\hb@xt@\z@{%
6230          \@defaultunitsset\@tempdimc{#1}\unitlength
6231          \kern\@tempdimc
6232          {\ifnum\bbl@tempc>\z@\bbl@pictresetdir\fi#3}\hss}%
6233        \ignorespaces}%
6234      \MakeRobust\put}%
6235    \AtBeginDocument
```

```
6236      {\AddToHook{cmd/diagbox@pict/before}{\let\bbl@pictsetdir\@gobble}%
6237       \ifx\pgfpicture\@undefined\else % TODO. Allow deactivate?
6238          \AddToHook{env/pgfpicture/begin}{\bbl@pictsetdir\@ne}%
6239          \bbl@add\pgfinterruptpicture{\bbl@pictresetdir}%
6240          \bbl@add\pgfsys@beginpicture{\bbl@pictsetdir\z@}%
6241       \fi
6242       \ifx\tikzpicture\@undefined\else
6243          \AddToHook{env/tikzpicture/begin}{\bbl@pictsetdir\z@}%
6244          \bbl@add\tikz@atbegin@node{\bbl@pictresetdir}%
6245          \bbl@sreplace\tikz{\begingroup}{\begingroup\bbl@pictsetdir\tw@}%
6246       \fi
6247       \ifx\tcolorbox\@undefined\else
6248          \AddToHook{env/tcolorbox/begin}{\bbl@pictsetdir\@ne}%
6249          \bbl@sreplace\tcb@savebox
6250            {\ignorespaces}{\ignorespaces\bbl@pictresetdir}%
6251          \ifx\tikzpicture@tcb@hooked\@undefined\else
6252            \bbl@sreplace\tikzpicture@tcb@hooked{\noexpand\tikzpicture}%
6253              {\textdir TLT\noexpand\tikzpicture}%
6254          \fi
6255       \fi
6256     }}
6257   {}
```

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.

```
6258 \IfBabelLayout{counters}%
6259   {\let\bbl@OL@@textsuperscript\@textsuperscript
6260    \bbl@sreplace\@textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
6261    \let\bbl@latinarabic=\@arabic
6262    \let\bbl@OL@@arabic\@arabic
6263    \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
6264    \@ifpackagewith{babel}{bidi=default}%
6265      {\let\bbl@asciiroman=\@roman
6266       \let\bbl@OL@@roman\@roman
6267       \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciiroman#1}}}%
6268       \let\bbl@asciiRoman=\@Roman
6269       \let\bbl@OL@@roman\@Roman
6270       \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}%
6271       \let\bbl@OL@labelenumii\labelenumii
6272       \def\labelenumii)}{\theenumii(}%
6273       \let\bbl@OL@p@enumiii\p@enumiii
6274       \def\p@enumiii{\p@enumii)}{\theenumii(}}{}}{}
6275 ⟨⟨Footnote changes⟩⟩
6276 \IfBabelLayout{footnotes}%
6277   {\let\bbl@OL@footnote\footnote
6278    \BabelFootnote\footnote\languagename{}{}%
6279    \BabelFootnote\localfootnote\languagename{}{}%
6280    \BabelFootnote\mainfootnote{}{}{}}
6281   {}
```

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.

```
6282 \IfBabelLayout{extras}%
6283   {\let\bbl@OL@underline\underline
6284    \bbl@sreplace\underline{$\@@underline}{\bbl@nextfake$\@@underline}%
6285    \let\bbl@OL@LaTeX2e\LaTeX2e
6286    \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
6287      \if b\expandafter\@car\f@series\@nil\boldmath\fi
6288      \babelsublr{%
6289        \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}}}
6290   {}
6291 ⟨/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.

```
6292 ⟨∗transforms⟩
6293 Babel.linebreaking.replacements = {}
6294 Babel.linebreaking.replacements[0] = {}  -- pre
6295 Babel.linebreaking.replacements[1] = {}  -- post
6296
6297 -- Discretionaries contain strings as nodes
6298 function Babel.str_to_nodes(fn, matches, base)
6299   local n, head, last
6300   if fn == nil then return nil end
6301   for s in string.utfvalues(fn(matches)) do
6302     if base.id == 7 then
6303       base = base.replace
6304     end
6305     n = node.copy(base)
6306     n.char     = s
6307     if not head then
6308       head = n
6309     else
6310       last.next = n
6311     end
6312     last = n
6313   end
6314   return head
6315 end
6316
6317 Babel.fetch_subtext = {}
6318
6319 Babel.ignore_pre_char = function(node)
6320   return (node.lang == Babel.nohyphenation)
6321 end
6322
6323 -- Merging both functions doesn't seen feasible, because there are too
6324 -- many differences.
6325 Babel.fetch_subtext[0] = function(head)
6326   local word_string = ''
6327   local word_nodes = {}
6328   local lang
6329   local item = head
6330   local inmath = false
6331
6332   while item do
6333
6334     if item.id == 11 then
6335       inmath = (item.subtype == 0)
6336     end
6337
6338     if inmath then
6339       -- pass
6340
6341     elseif item.id == 29 then
```

```lua
6342        local locale = node.get_attribute(item, Babel.attr_locale)
6343
6344        if lang == locale or lang == nil then
6345          lang = lang or locale
6346          if Babel.ignore_pre_char(item) then
6347            word_string = word_string .. Babel.us_char
6348          else
6349            word_string = word_string .. unicode.utf8.char(item.char)
6350          end
6351          word_nodes[#word_nodes+1] = item
6352        else
6353          break
6354        end
6355
6356      elseif item.id == 12 and item.subtype == 13 then
6357        word_string = word_string .. ' '
6358        word_nodes[#word_nodes+1] = item
6359
6360      -- Ignore leading unrecognized nodes, too.
6361      elseif word_string ~= '' then
6362        word_string = word_string .. Babel.us_char
6363        word_nodes[#word_nodes+1] = item  -- Will be ignored
6364      end
6365
6366      item = item.next
6367    end
6368
6369    -- Here and above we remove some trailing chars but not the
6370    -- corresponding nodes. But they aren't accessed.
6371    if word_string:sub(-1) == ' ' then
6372      word_string = word_string:sub(1,-2)
6373    end
6374    word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
6375    return word_string, word_nodes, item, lang
6376 end
6377
6378 Babel.fetch_subtext[1] = function(head)
6379    local word_string = ''
6380    local word_nodes = {}
6381    local lang
6382    local item = head
6383    local inmath = false
6384
6385    while item do
6386
6387      if item.id == 11 then
6388        inmath = (item.subtype == 0)
6389      end
6390
6391      if inmath then
6392        -- pass
6393
6394      elseif item.id == 29 then
6395        if item.lang == lang or lang == nil then
6396          if (item.char ~= 124) and (item.char ~= 61) then -- not =, not |
6397            lang = lang or item.lang
6398            word_string = word_string .. unicode.utf8.char(item.char)
6399            word_nodes[#word_nodes+1] = item
6400          end
6401        else
6402          break
6403        end
6404
```

```
6405      elseif item.id == 7 and item.subtype == 2 then
6406        word_string = word_string .. '='
6407        word_nodes[#word_nodes+1] = item
6408
6409      elseif item.id == 7 and item.subtype == 3 then
6410        word_string = word_string .. '|'
6411        word_nodes[#word_nodes+1] = item
6412
6413      -- (1) Go to next word if nothing was found, and (2) implicitly
6414      -- remove leading USs.
6415      elseif word_string == '' then
6416        -- pass
6417
6418      -- This is the responsible for splitting by words.
6419      elseif (item.id == 12 and item.subtype == 13) then
6420        break
6421
6422      else
6423        word_string = word_string .. Babel.us_char
6424        word_nodes[#word_nodes+1] = item  -- Will be ignored
6425      end
6426
6427      item = item.next
6428    end
6429
6430    word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
6431    return word_string, word_nodes, item, lang
6432 end
6433
6434 function Babel.pre_hyphenate_replace(head)
6435    Babel.hyphenate_replace(head, 0)
6436 end
6437
6438 function Babel.post_hyphenate_replace(head)
6439    Babel.hyphenate_replace(head, 1)
6440 end
6441
6442 Babel.us_char = string.char(31)
6443
6444 function Babel.hyphenate_replace(head, mode)
6445    local u = unicode.utf8
6446    local lbkr = Babel.linebreaking.replacements[mode]
6447
6448    local word_head = head
6449
6450    while true do  -- for each subtext block
6451
6452      local w, w_nodes, nw, lang = Babel.fetch_subtext[mode](word_head)
6453
6454      if Babel.debug then
6455        print()
6456        print((mode == 0) and '@@@@<' or '@@@@>', w)
6457      end
6458
6459      if nw == nil and w == '' then break end
6460
6461      if not lang then goto next end
6462      if not lbkr[lang] then goto next end
6463
6464      -- For each saved (pre|post)hyphenation. TODO. Reconsider how
6465      -- loops are nested.
6466      for k=1, #lbkr[lang] do
6467        local p = lbkr[lang][k].pattern
```

```
6468        local r = lbkr[lang][k].replace
6469        local attr = lbkr[lang][k].attr or -1
6470
6471        if Babel.debug then
6472          print('*****', p, mode)
6473        end
6474
6475        -- This variable is set in some cases below to the first *byte*
6476        -- after the match, either as found by u.match (faster) or the
6477        -- computed position based on sc if w has changed.
6478        local last_match = 0
6479        local step = 0
6480
6481        -- For every match.
6482        while true do
6483          if Babel.debug then
6484            print('=====')
6485          end
6486          local new  -- used when inserting and removing nodes
6487
6488          local matches = { u.match(w, p, last_match) }
6489
6490          if #matches < 2 then break end
6491
6492          -- Get and remove empty captures (with ()'s, which return a
6493          -- number with the position), and keep actual captures
6494          -- (from (...)), if any, in matches.
6495          local first = table.remove(matches, 1)
6496          local last  = table.remove(matches, #matches)
6497          -- Non re-fetched substrings may contain \31, which separates
6498          -- subsubstrings.
6499          if string.find(w:sub(first, last-1), Babel.us_char) then break end
6500
6501          local save_last = last -- with A()BC()D, points to D
6502
6503          -- Fix offsets, from bytes to unicode. Explained above.
6504          first = u.len(w:sub(1, first-1)) + 1
6505          last  = u.len(w:sub(1, last-1)) -- now last points to C
6506
6507          -- This loop stores in a small table the nodes
6508          -- corresponding to the pattern. Used by 'data' to provide a
6509          -- predictable behavior with 'insert' (w_nodes is modified on
6510          -- the fly), and also access to 'remove'd nodes.
6511          local sc = first-1              -- Used below, too
6512          local data_nodes = {}
6513
6514          local enabled = true
6515          for q = 1, last-first+1 do
6516            data_nodes[q] = w_nodes[sc+q]
6517            if enabled
6518               and attr > -1
6519               and not node.has_attribute(data_nodes[q], attr)
6520            then
6521              enabled = false
6522            end
6523          end
6524
6525          -- This loop traverses the matched substring and takes the
6526          -- corresponding action stored in the replacement list.
6527          -- sc = the position in substr nodes / string
6528          -- rc = the replacement table index
6529          local rc = 0
6530
```

```
6531          while rc < last-first+1 do -- for each replacement
6532            if Babel.debug then
6533              print('.....', rc + 1)
6534            end
6535            sc = sc + 1
6536            rc = rc + 1
6537
6538            if Babel.debug then
6539              Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6540              local ss = ''
6541              for itt in node.traverse(head) do
6542               if itt.id == 29 then
6543                 ss = ss .. unicode.utf8.char(itt.char)
6544               else
6545                 ss = ss .. '{' .. itt.id .. '}'
6546               end
6547              end
6548              print('*****************', ss)
6549
6550            end
6551
6552            local crep = r[rc]
6553            local item = w_nodes[sc]
6554            local item_base = item
6555            local placeholder = Babel.us_char
6556            local d
6557
6558            if crep and crep.data then
6559              item_base = data_nodes[crep.data]
6560            end
6561
6562            if crep then
6563              step = crep.step or 0
6564            end
6565
6566            if (not enabled) or (crep and next(crep) == nil) then -- = {}
6567              last_match = save_last    -- Optimization
6568              goto next
6569
6570            elseif crep == nil or crep.remove then
6571              node.remove(head, item)
6572              table.remove(w_nodes, sc)
6573              w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
6574              sc = sc - 1  -- Nothing has been inserted.
6575              last_match = utf8.offset(w, sc+1+step)
6576              goto next
6577
6578            elseif crep and crep.kashida then -- Experimental
6579              node.set_attribute(item,
6580                  Babel.attr_kashida,
6581                  crep.kashida)
6582              last_match = utf8.offset(w, sc+1+step)
6583              goto next
6584
6585            elseif crep and crep.string then
6586              local str = crep.string(matches)
6587              if str == '' then  -- Gather with nil
6588                node.remove(head, item)
6589                table.remove(w_nodes, sc)
6590                w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
6591                sc = sc - 1  -- Nothing has been inserted.
6592              else
6593                local loop_first = true
```

190

```
6594                for s in string.utfvalues(str) do
6595                  d = node.copy(item_base)
6596                  d.char = s
6597                  if loop_first then
6598                    loop_first = false
6599                    head, new = node.insert_before(head, item, d)
6600                    if sc == 1 then
6601                      word_head = head
6602                    end
6603                    w_nodes[sc] = d
6604                    w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc+1)
6605                  else
6606                    sc = sc + 1
6607                    head, new = node.insert_before(head, item, d)
6608                    table.insert(w_nodes, sc, new)
6609                    w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc)
6610                  end
6611                  if Babel.debug then
6612                    print('.....', 'str')
6613                    Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6614                  end
6615                end  -- for
6616                node.remove(head, item)
6617              end  -- if ''
6618              last_match = utf8.offset(w, sc+1+step)
6619              goto next
6620
6621          elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
6622              d = node.new(7, 0)   -- (disc, discretionary)
6623              d.pre    = Babel.str_to_nodes(crep.pre, matches, item_base)
6624              d.post   = Babel.str_to_nodes(crep.post, matches, item_base)
6625              d.replace = Babel.str_to_nodes(crep.no, matches, item_base)
6626              d.attr = item_base.attr
6627              if crep.pre == nil then  -- TeXbook p96
6628                d.penalty = crep.penalty or tex.hyphenpenalty
6629              else
6630                d.penalty = crep.penalty or tex.exhyphenpenalty
6631              end
6632              placeholder = '|'
6633              head, new = node.insert_before(head, item, d)
6634
6635          elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
6636              -- ERROR
6637
6638          elseif crep and crep.penalty then
6639              d = node.new(14, 0)   -- (penalty, userpenalty)
6640              d.attr = item_base.attr
6641              d.penalty = crep.penalty
6642              head, new = node.insert_before(head, item, d)
6643
6644          elseif crep and crep.space then
6645              -- 655360 = 10 pt = 10 * 65536 sp
6646              d = node.new(12, 13)      -- (glue, spaceskip)
6647              local quad = font.getfont(item_base.font).size or 655360
6648              node.setglue(d, crep.space[1] * quad,
6649                              crep.space[2] * quad,
6650                              crep.space[3] * quad)
6651              if mode == 0 then
6652                placeholder = ' '
6653              end
6654              head, new = node.insert_before(head, item, d)
6655
6656          elseif crep and crep.spacefactor then
```

191

```
6657            d = node.new(12, 13)        -- (glue, spaceskip)
6658            local base_font = font.getfont(item_base.font)
6659            node.setglue(d,
6660              crep.spacefactor[1] * base_font.parameters['space'],
6661              crep.spacefactor[2] * base_font.parameters['space_stretch'],
6662              crep.spacefactor[3] * base_font.parameters['space_shrink'])
6663            if mode == 0 then
6664              placeholder = ' '
6665            end
6666            head, new = node.insert_before(head, item, d)
6667
6668          elseif mode == 0 and crep and crep.space then
6669            -- ERROR
6670
6671          end  -- ie replacement cases
6672
6673          -- Shared by disc, space and penalty.
6674          if sc == 1 then
6675            word_head = head
6676          end
6677          if crep.insert then
6678            w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc)
6679            table.insert(w_nodes, sc, new)
6680            last = last + 1
6681          else
6682            w_nodes[sc] = d
6683            node.remove(head, item)
6684            w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1)
6685          end
6686
6687          last_match = utf8.offset(w, sc+1+step)
6688
6689          ::next::
6690
6691        end  -- for each replacement
6692
6693        if Babel.debug then
6694            print('.....', '/')
6695            Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6696        end
6697
6698      end  -- for match
6699
6700    end  -- for patterns
6701
6702    ::next::
6703    word_head = nw
6704  end  -- for substring
6705  return head
6706 end
6707
6708 -- This table stores capture maps, numbered consecutively
6709 Babel.capture_maps = {}
6710
6711 -- The following functions belong to the next macro
6712 function Babel.capture_func(key, cap)
6713  local ret = "[[" .. cap:gsub('{([0-9])}', "]]..m[%1]..[[") .. "]]"
6714  local cnt
6715  local u = unicode.utf8
6716  ret, cnt = ret:gsub('{([0-9])}|([^|]+)|(.-)}', Babel.capture_func_map)
6717  if cnt == 0 then
6718    ret = u.gsub(ret, '{(%x%x%x%x+)}',
6719          function (n)
```

```
6720             return u.char(tonumber(n, 16))
6721         end)
6722   end
6723   ret = ret:gsub("%[%[%]%]%.%.", '')
6724   ret = ret:gsub("%.%.%[%[%]%]", '')
6725   return key .. [[=function(m) return ]] .. ret .. [[ end]]
6726 end
6727
6728 function Babel.capt_map(from, mapno)
6729   return Babel.capture_maps[mapno][from] or from
6730 end
6731
6732 -- Handle the {n|abc|ABC} syntax in captures
6733 function Babel.capture_func_map(capno, from, to)
6734   local u = unicode.utf8
6735   from = u.gsub(from, '{(%x%x%x%x+)}',
6736         function (n)
6737           return u.char(tonumber(n, 16))
6738         end)
6739   to = u.gsub(to, '{(%x%x%x%x+)}',
6740         function (n)
6741           return u.char(tonumber(n, 16))
6742         end)
6743   local froms = {}
6744   for s in string.utfcharacters(from) do
6745     table.insert(froms, s)
6746   end
6747   local cnt = 1
6748   table.insert(Babel.capture_maps, {})
6749   local mlen = table.getn(Babel.capture_maps)
6750   for s in string.utfcharacters(to) do
6751     Babel.capture_maps[mlen][froms[cnt]] = s
6752     cnt = cnt + 1
6753   end
6754   return "]]..Babel.capt_map(m[" .. capno .. "]," ..
6755         (mlen) .. ")..".. "[["
6756 end
6757
6758 -- Create/Extend reversed sorted list of kashida weights:
6759 function Babel.capture_kashida(key, wt)
6760   wt = tonumber(wt)
6761   if Babel.kashida_wts then
6762     for p, q in ipairs(Babel.kashida_wts) do
6763       if wt  == q then
6764         break
6765       elseif wt > q then
6766         table.insert(Babel.kashida_wts, p, wt)
6767         break
6768       elseif table.getn(Babel.kashida_wts) == p then
6769         table.insert(Babel.kashida_wts, wt)
6770       end
6771     end
6772   else
6773     Babel.kashida_wts = { wt }
6774   end
6775   return 'kashida = ' .. wt
6776 end
6777 ⟨/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.

```
6778 ⟨*basic-r⟩
6779 Babel = Babel or {}
6780
6781 Babel.bidi_enabled = true
6782
6783 require('babel-data-bidi.lua')
6784
6785 local characters = Babel.characters
6786 local ranges = Babel.ranges
6787
6788 local DIR = node.id("dir")
6789
6790 local function dir_mark(head, from, to, outer)
6791   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
6792   local d = node.new(DIR)
6793   d.dir = '+' .. dir
6794   node.insert_before(head, from, d)
6795   d = node.new(DIR)
6796   d.dir = '-' .. dir
6797   node.insert_after(head, to, d)
6798 end
6799
6800 function Babel.bidi(head, ispar)
6801   local first_n, last_n          -- first and last char with nums
6802   local last_es                  -- an auxiliary 'last' used with nums
6803   local first_d, last_d          -- first and last char in L/R block
6804   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):

```
6805    local strong = ('TRT' == tex.pardir) and 'r' or 'l'
6806    local strong_lr = (strong == 'l') and 'l' or 'r'
6807    local outer = strong
6808
6809    local new_dir = false
6810    local first_dir = false
6811    local inmath = false
6812
6813    local last_lr
6814
6815    local type_n = ''
6816
6817    for item in node.traverse(head) do
6818
6819      -- three cases: glyph, dir, otherwise
6820      if item.id == node.id'glyph'
6821        or (item.id == 7 and item.subtype == 2) then
6822
6823        local itemchar
6824        if item.id == 7 and item.subtype == 2 then
6825          itemchar = item.replace.char
6826        else
6827          itemchar = item.char
6828        end
6829        local chardata = characters[itemchar]
6830        dir = chardata and chardata.d or nil
6831        if not dir then
6832          for nn, et in ipairs(ranges) do
6833            if itemchar < et[1] then
6834              break
6835            elseif itemchar <= et[2] then
6836              dir = et[3]
6837              break
6838            end
6839          end
6840        end
6841        dir = dir or 'l'
6842        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).

```
6843        if new_dir then
6844          attr_dir = 0
6845          for at in node.traverse(item.attr) do
6846            if at.number == Babel.attr_dir then
6847              attr_dir = at.value % 3
6848            end
6849          end
6850          if attr_dir == 1 then
6851            strong = 'r'
6852          elseif attr_dir == 2 then
6853            strong = 'al'
6854          else
6855            strong = 'l'
6856          end
6857          strong_lr = (strong == 'l') and 'l' or 'r'
6858          outer = strong_lr
6859          new_dir = false
6860        end
6861
```

```
6862        if dir == 'nsm' then dir = strong end              -- W1
```

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

```
6863        dir_real = dir                    -- We need dir_real to set strong below
6864        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:

```
6865        if strong == 'al' then
6866          if dir == 'en' then dir = 'an' end              -- W2
6867          if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
6868          strong_lr = 'r'                                -- W3
6869        end
```

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

```
6870      elseif item.id == node.id'dir' and not inmath then
6871        new_dir = true
6872        dir = nil
6873      elseif item.id == node.id'math' then
6874        inmath = (item.subtype == 0)
6875      else
6876        dir = nil           -- Not a char
6877      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>.

```
6878        if dir == 'en' or dir == 'an' or dir == 'et' then
6879          if dir ~= 'et' then
6880            type_n = dir
6881          end
6882          first_n = first_n or item
6883          last_n = last_es or item
6884          last_es = nil
6885        elseif dir == 'es' and last_n then -- W3+W6
6886          last_es = item
6887        elseif dir == 'cs' then               -- it's right - do nothing
6888        elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
6889          if strong_lr == 'r' and type_n ~= '' then
6890            dir_mark(head, first_n, last_n, 'r')
6891          elseif strong_lr == 'l' and first_d and type_n == 'an' then
6892            dir_mark(head, first_n, last_n, 'r')
6893            dir_mark(head, first_d, last_d, outer)
6894            first_d, last_d = nil, nil
6895          elseif strong_lr == 'l' and type_n ~= '' then
6896            last_d = last_n
6897          end
6898          type_n = ''
6899          first_n, last_n = nil, nil
6900        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:

```
6901        if dir == 'l' or dir == 'r' then
6902          if dir ~= outer then
6903            first_d = first_d or item
6904            last_d = item
6905          elseif first_d and dir ~= strong_lr then
6906            dir_mark(head, first_d, last_d, outer)
6907            first_d, last_d = nil, nil
```

```
6908      end
6909    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.

```
6910    if dir and not last_lr and dir ~= 'l' and outer == 'r' then
6911      item.char = characters[item.char] and
6912                  characters[item.char].m or item.char
6913    elseif (dir or new_dir) and last_lr ~= item then
6914      local mir = outer .. strong_lr .. (dir or outer)
6915      if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
6916        for ch in node.traverse(node.next(last_lr)) do
6917          if ch == item then break end
6918          if ch.id == node.id'glyph' and characters[ch.char] then
6919            ch.char = characters[ch.char].m or ch.char
6920          end
6921        end
6922      end
6923    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).

```
6924    if dir == 'l' or dir == 'r' then
6925      last_lr = item
6926      strong = dir_real            -- Don't search back - best save now
6927      strong_lr = (strong == 'l') and 'l' or 'r'
6928    elseif new_dir then
6929      last_lr = nil
6930    end
6931  end
```

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

```
6932  if last_lr and outer == 'r' then
6933    for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
6934      if characters[ch.char] then
6935        ch.char = characters[ch.char].m or ch.char
6936      end
6937    end
6938  end
6939  if first_n then
6940    dir_mark(head, first_n, last_n, outer)
6941  end
6942  if first_d then
6943    dir_mark(head, first_d, last_d, outer)
6944  end
```

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

```
6945  return node.prev(head) or head
6946 end
6947 ⟨/basic-r⟩
```

And here the Lua code for bidi=basic:

```
6948 ⟨*basic⟩
6949 Babel = Babel or {}
6950
6951 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
6952
6953 Babel.fontmap = Babel.fontmap or {}
6954 Babel.fontmap[0] = {}      -- l
6955 Babel.fontmap[1] = {}      -- r
```

```lua
6956 Babel.fontmap[2] = {}        -- al/an
6957
6958 Babel.bidi_enabled = true
6959 Babel.mirroring_enabled = true
6960
6961 require('babel-data-bidi.lua')
6962
6963 local characters = Babel.characters
6964 local ranges = Babel.ranges
6965
6966 local DIR = node.id('dir')
6967 local GLYPH = node.id('glyph')
6968
6969 local function insert_implicit(head, state, outer)
6970   local new_state = state
6971   if state.sim and state.eim and state.sim ~= state.eim then
6972     dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
6973     local d = node.new(DIR)
6974     d.dir = '+' .. dir
6975     node.insert_before(head, state.sim, d)
6976     local d = node.new(DIR)
6977     d.dir = '-' .. dir
6978     node.insert_after(head, state.eim, d)
6979   end
6980   new_state.sim, new_state.eim = nil, nil
6981   return head, new_state
6982 end
6983
6984 local function insert_numeric(head, state)
6985   local new
6986   local new_state = state
6987   if state.san and state.ean and state.san ~= state.ean then
6988     local d = node.new(DIR)
6989     d.dir = '+TLT'
6990     _, new = node.insert_before(head, state.san, d)
6991     if state.san == state.sim then state.sim = new end
6992     local d = node.new(DIR)
6993     d.dir = '-TLT'
6994     _, new = node.insert_after(head, state.ean, d)
6995     if state.ean == state.eim then state.eim = new end
6996   end
6997   new_state.san, new_state.ean = nil, nil
6998   return head, new_state
6999 end
7000
7001 -- TODO - \hbox with an explicit dir can lead to wrong results
7002 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
7003 -- was s made to improve the situation, but the problem is the 3-dir
7004 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
7005 -- well.
7006
7007 function Babel.bidi(head, ispar, hdir)
7008   local d    -- d is used mainly for computations in a loop
7009   local prev_d = ''
7010   local new_d = false
7011
7012   local nodes = {}
7013   local outer_first = nil
7014   local inmath = false
7015
7016   local glue_d = nil
7017   local glue_i = nil
7018
```

```
7019  local has_en = false
7020  local first_et = nil
7021
7022  local ATDIR = Babel.attr_dir
7023
7024  local save_outer
7025  local temp = node.get_attribute(head, ATDIR)
7026  if temp then
7027    temp = temp % 3
7028    save_outer = (temp == 0 and 'l') or
7029                 (temp == 1 and 'r') or
7030                 (temp == 2 and 'al')
7031  elseif ispar then              -- Or error? Shouldn't happen
7032    save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
7033  else                           -- Or error? Shouldn't happen
7034    save_outer = ('TRT' == hdir) and 'r' or 'l'
7035  end
7036    -- when the callback is called, we are just _after_ the box,
7037    -- and the textdir is that of the surrounding text
7038  -- if not ispar and hdir ~= tex.textdir then
7039  --    save_outer = ('TRT' == hdir) and 'r' or 'l'
7040  -- end
7041  local outer = save_outer
7042  local last = outer
7043  -- 'al' is only taken into account in the first, current loop
7044  if save_outer == 'al' then save_outer = 'r' end
7045
7046  local fontmap = Babel.fontmap
7047
7048  for item in node.traverse(head) do
7049
7050    -- In what follows, #node is the last (previous) node, because the
7051    -- current one is not added until we start processing the neutrals.
7052
7053    -- three cases: glyph, dir, otherwise
7054    if item.id == GLYPH
7055       or (item.id == 7 and item.subtype == 2) then
7056
7057      local d_font = nil
7058      local item_r
7059      if item.id == 7 and item.subtype == 2 then
7060        item_r = item.replace    -- automatic discs have just 1 glyph
7061      else
7062        item_r = item
7063      end
7064      local chardata = characters[item_r.char]
7065      d = chardata and chardata.d or nil
7066      if not d or d == 'nsm' then
7067        for nn, et in ipairs(ranges) do
7068          if item_r.char < et[1] then
7069            break
7070          elseif item_r.char <= et[2] then
7071            if not d then d = et[3]
7072            elseif d == 'nsm' then d_font = et[3]
7073            end
7074            break
7075          end
7076        end
7077      end
7078      d = d or 'l'
7079
7080      -- A short 'pause' in bidi for mapfont
7081      d_font = d_font or d
```

```
7082       d_font = (d_font == 'l' and 0) or
7083                (d_font == 'nsm' and 0) or
7084                (d_font == 'r' and 1) or
7085                (d_font == 'al' and 2) or
7086                (d_font == 'an' and 2) or nil
7087       if d_font and fontmap and fontmap[d_font][item_r.font] then
7088         item_r.font = fontmap[d_font][item_r.font]
7089       end
7090
7091       if new_d then
7092         table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
7093         if inmath then
7094           attr_d = 0
7095         else
7096           attr_d = node.get_attribute(item, ATDIR)
7097           attr_d = attr_d % 3
7098         end
7099         if attr_d == 1 then
7100           outer_first = 'r'
7101           last = 'r'
7102         elseif attr_d == 2 then
7103           outer_first = 'r'
7104           last = 'al'
7105         else
7106           outer_first = 'l'
7107           last = 'l'
7108         end
7109         outer = last
7110         has_en = false
7111         first_et = nil
7112         new_d = false
7113       end
7114
7115       if glue_d then
7116         if (d == 'l' and 'l' or 'r') ~= glue_d then
7117             table.insert(nodes, {glue_i, 'on', nil})
7118         end
7119         glue_d = nil
7120         glue_i = nil
7121       end
7122
7123    elseif item.id == DIR then
7124      d = nil
7125      if head ~= item then new_d = true end
7126
7127    elseif item.id == node.id'glue' and item.subtype == 13 then
7128      glue_d = d
7129      glue_i = item
7130      d = nil
7131
7132    elseif item.id == node.id'math' then
7133      inmath = (item.subtype == 0)
7134
7135    else
7136      d = nil
7137    end
7138
7139    -- AL <= EN/ET/ES      -- W2 + W3 + W6
7140    if last == 'al' and d == 'en' then
7141      d = 'an'            -- W3
7142    elseif last == 'al' and (d == 'et' or d == 'es') then
7143      d = 'on'            -- W6
7144    end
```

```
7145
7146    -- EN + CS/ES + EN      -- W4
7147    if d == 'en' and #nodes >= 2 then
7148      if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
7149          and nodes[#nodes-1][2] == 'en' then
7150        nodes[#nodes][2] = 'en'
7151      end
7152    end
7153
7154    -- AN + CS + AN          -- W4 too, because uax9 mixes both cases
7155    if d == 'an' and #nodes >= 2 then
7156      if (nodes[#nodes][2] == 'cs')
7157          and nodes[#nodes-1][2] == 'an' then
7158        nodes[#nodes][2] = 'an'
7159      end
7160    end
7161
7162    -- ET/EN                 -- W5 + W7->l / W6->on
7163    if d == 'et' then
7164      first_et = first_et or (#nodes + 1)
7165    elseif d == 'en' then
7166      has_en = true
7167      first_et = first_et or (#nodes + 1)
7168    elseif first_et then        -- d may be nil here !
7169      if has_en then
7170        if last == 'l' then
7171          temp = 'l'     -- W7
7172        else
7173          temp = 'en'    -- W5
7174        end
7175      else
7176        temp = 'on'      -- W6
7177      end
7178      for e = first_et, #nodes do
7179        if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
7180      end
7181      first_et = nil
7182      has_en = false
7183    end
7184
7185    -- Force mathdir in math if ON (currently works as expected only
7186    -- with 'l')
7187    if inmath and d == 'on' then
7188      d = ('TRT' == tex.mathdir) and 'r' or 'l'
7189    end
7190
7191    if d then
7192      if d == 'al' then
7193        d = 'r'
7194        last = 'al'
7195      elseif d == 'l' or d == 'r' then
7196        last = d
7197      end
7198      prev_d = d
7199      table.insert(nodes, {item, d, outer_first})
7200    end
7201
7202    outer_first = nil
7203
7204  end
7205
7206  -- TODO -- repeated here in case EN/ET is the last node. Find a
7207  -- better way of doing things:
```

```
7208  if first_et then        -- dir may be nil here !
7209    if has_en then
7210      if last == 'l' then
7211        temp = 'l'     -- W7
7212      else
7213        temp = 'en'    -- W5
7214      end
7215    else
7216      temp = 'on'      -- W6
7217    end
7218    for e = first_et, #nodes do
7219      if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
7220    end
7221  end
7222
7223  -- dummy node, to close things
7224  table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
7225
7226  --------------   NEUTRAL  ----------------
7227
7228  outer = save_outer
7229  last = outer
7230
7231  local first_on = nil
7232
7233  for q = 1, #nodes do
7234    local item
7235
7236    local outer_first = nodes[q][3]
7237    outer = outer_first or outer
7238    last = outer_first or last
7239
7240    local d = nodes[q][2]
7241    if d == 'an' or d == 'en' then d = 'r' end
7242    if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
7243
7244    if d == 'on' then
7245      first_on = first_on or q
7246    elseif first_on then
7247      if last == d then
7248        temp = d
7249      else
7250        temp = outer
7251      end
7252      for r = first_on, q - 1 do
7253        nodes[r][2] = temp
7254        item = nodes[r][1]     -- MIRRORING
7255        if Babel.mirroring_enabled and item.id == GLYPH
7256            and temp == 'r' and characters[item.char] then
7257          local font_mode = ''
7258          if font.fonts[item.font].properties then
7259            font_mode = font.fonts[item.font].properties.mode
7260          end
7261          if font_mode ~= 'harf' and font_mode ~= 'plug' then
7262            item.char = characters[item.char].m or item.char
7263          end
7264        end
7265      end
7266      first_on = nil
7267    end
7268
7269    if d == 'r' or d == 'l' then last = d end
7270  end
```

```
7271
7272  -------------  IMPLICIT, REORDER ---------------
7273
7274  outer = save_outer
7275  last = outer
7276
7277  local state = {}
7278  state.has_r = false
7279
7280  for q = 1, #nodes do
7281
7282    local item = nodes[q][1]
7283
7284    outer = nodes[q][3] or outer
7285
7286    local d = nodes[q][2]
7287
7288    if d == 'nsm' then d = last end            -- W1
7289    if d == 'en' then d = 'an' end
7290    local isdir = (d == 'r' or d == 'l')
7291
7292    if outer == 'l' and d == 'an' then
7293      state.san = state.san or item
7294      state.ean = item
7295    elseif state.san then
7296      head, state = insert_numeric(head, state)
7297    end
7298
7299    if outer == 'l' then
7300      if d == 'an' or d == 'r' then      -- im -> implicit
7301        if d == 'r' then state.has_r = true end
7302        state.sim = state.sim or item
7303        state.eim = item
7304      elseif d == 'l' and state.sim and state.has_r then
7305        head, state = insert_implicit(head, state, outer)
7306      elseif d == 'l' then
7307        state.sim, state.eim, state.has_r = nil, nil, false
7308      end
7309    else
7310      if d == 'an' or d == 'l' then
7311        if nodes[q][3] then -- nil except after an explicit dir
7312          state.sim = item  -- so we move sim 'inside' the group
7313        else
7314          state.sim = state.sim or item
7315        end
7316        state.eim = item
7317      elseif d == 'r' and state.sim then
7318        head, state = insert_implicit(head, state, outer)
7319      elseif d == 'r' then
7320        state.sim, state.eim = nil, nil
7321      end
7322    end
7323
7324    if isdir then
7325      last = d            -- Don't search back - best save now
7326    elseif d == 'on' and state.san  then
7327      state.san = state.san or item
7328      state.ean = item
7329    end
7330
7331  end
7332
7333  return node.prev(head) or head
```

```
7334 end
7335 ⟨/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.

```
7336 ⟨∗nil⟩
7337 \ProvidesLanguage{nil}[⟨⟨date⟩⟩ ⟨⟨version⟩⟩ Nil language]
7338 \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.

```
7339 \ifx\l@nil\@undefined
7340   \newlanguage\l@nil
7341   \@namedef{bbl@hyphendata@\the\l@nil}{{}{}}% Remove warning
7342   \let\bbl@elt\relax
7343   \edef\bbl@languages{%  Add it to the list of languages
7344     \bbl@languages\bbl@elt{nil}{\the\l@nil}{}{}}
7345 \fi
```

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

```
7346 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}
```

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

\captionnil
\datenil
```
7347 \let\captionsnil\@empty
7348 \let\datenil\@empty
```

There is no locale file for this pseudo-language, so the corresponding fields are defined here.

```
7349 \def\bbl@inidata@nil{%
7350   \bbl@elt{identification}{tag.ini}{und}%
7351   \bbl@elt{identification}{load.level}{0}%
7352   \bbl@elt{identification}{charset}{utf8}%
7353   \bbl@elt{identification}{version}{1.0}%
7354   \bbl@elt{identification}{date}{2022-05-16}%
7355   \bbl@elt{identification}{name.local}{nil}%
7356   \bbl@elt{identification}{name.english}{nil}%
7357   \bbl@elt{identification}{name.babel}{nil}%
7358   \bbl@elt{identification}{tag.bcp47}{und}%
7359   \bbl@elt{identification}{language.tag.bcp47}{und}%
7360   \bbl@elt{identification}{tag.opentype}{dflt}%
7361   \bbl@elt{identification}{script.name}{Latin}%
7362   \bbl@elt{identification}{script.tag.bcp47}{Latn}%
```

```
7363    \bbl@elt{identification}{script.tag.opentype}{DFLT}%
7364    \bbl@elt{identification}{level}{1}%
7365    \bbl@elt{identification}{encodings}{}%
7366    \bbl@elt{identification}{derivate}{no}}
7367 \@namedef{bbl@tbcp@nil}{und}
7368 \@namedef{bbl@lbcp@nil}{und}
7369 \@namedef{bbl@lotf@nil}{dflt}
7370 \@namedef{bbl@elname@nil}{nil}
7371 \@namedef{bbl@lname@nil}{nil}
7372 \@namedef{bbl@esname@nil}{Latin}
7373 \@namedef{bbl@sname@nil}{Latin}
7374 \@namedef{bbl@sbcp@nil}{Latn}
7375 \@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.

```
7376 \ldf@finish{nil}
7377 ⟨/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.

## 15.1  Islamic

Start with function to compute the Julian day. It's based on the little library calendar.js, by John Walker, in the public domain. The code for the Civil calendar is based on it, too.

```
7378 ⟨*ca-islamic⟩
7379 \ExplSyntaxOn
7380 \def\bbl@fpmod#1#2{(#1-#2*floor(#1/#2))}
7381 \def\bbl@cs@gregleap#1{%
7382   (\bbl@fpmod{#1}{4} == 0) &&
7383     (!((\bbl@fpmod{#1}{100} == 0) && (\bbl@fpmod{#1}{400} != 0)))}
7384 \def\bbl@ca@jd#1#2#3{% year, month, day
7385   \fp_eval:n{ 1721424.5   + (365 * (#1 - 1)) +
7386     floor((#1 - 1) / 4)   + (-floor((#1 - 1) / 100)) +
7387     floor((#1 - 1) / 400) + floor((((367 * #2) - 362) / 12) +
7388     ((#2 <= 2) ? 0 : (\bbl@cs@gregleap{#1} ? -1 : -2)) + #3) }}
7389 % == islamic (default)
7390 % Not yet implemented
7391 \def\bbl@ca@islamic#1-#2-#3\@@#4#5#6{}
```

The Civil calendar.

```
7392 \def\bbl@cs@isltojd#1#2#3{ % year, month, day
7393   ((#3 + ceil(29.5 * (#2 - 1)) +
7394   (#1 - 1) * 354 + floor((3 + (11 * #1)) / 30) +
7395   1948439.5) - 1) }
7396 \@namedef{bbl@ca@islamic-civil++}{\bbl@ca@islamicvl@x{+2}}
7397 \@namedef{bbl@ca@islamic-civil+}{\bbl@ca@islamicvl@x{+1}}
7398 \@namedef{bbl@ca@islamic-civil}{\bbl@ca@islamicvl@x{}}
7399 \@namedef{bbl@ca@islamic-civil-}{\bbl@ca@islamicvl@x{-1}}
7400 \@namedef{bbl@ca@islamic-civil--}{\bbl@ca@islamicvl@x{-2}}
7401 \def\bbl@ca@islamicvl@x#1#2-#3-#4\@@#5#6#7{%
7402   \edef\bbl@tempa{%
7403     \fp_eval:n{ floor(\bbl@ca@jd{#2}{#3}{#4})+0.5 #1}}%
7404   \edef#5{%
7405     \fp_eval:n{ floor(((30*(\bbl@tempa-1948439.5)) + 10646)/10631) }}%
7406   \edef#6{\fp_eval:n{
7407     min(12,ceil((\bbl@tempa-(29+\bbl@cs@isltojd{#5}{1}{1}))/29.5)+1) }}%
7408   \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).

```
7409 \def\bbl@ca@umalqura@data{56660, 56690,56719,56749,56778,56808,%
7410   56837,56867,56897,56926,56956,56985,57015,57044,57074,57103,%
7411   57133,57162,57192,57221,57251,57280,57310,57340,57369,57399,%
7412   57429,57458,57487,57517,57546,57576,57605,57634,57664,57694,%
7413   57723,57753,57783,57813,57842,57871,57901,57930,57959,57989,%
7414   58018,58048,58077,58107,58137,58167,58196,58226,58255,58285,%
7415   58314,58343,58373,58402,58432,58461,58491,58521,58551,58580,%
7416   58610,58639,58669,58698,58727,58757,58786,58816,58845,58875,%
7417   58905,58934,58964,58994,59023,59053,59082,59111,59141,59170,%
7418   59200,59229,59259,59288,59318,59348,59377,59407,59436,59466,%
7419   59495,59525,59554,59584,59613,59643,59672,59702,59731,59761,%
7420   59791,59820,59850,59879,59909,59939,59968,59997,60027,60056,%
7421   60086,60115,60145,60174,60204,60234,60264,60293,60323,60352,%
7422   60381,60411,60440,60469,60499,60528,60558,60588,60618,60648,%
7423   60677,60707,60736,60765,60795,60824,60853,60883,60912,60942,%
7424   60972,61002,61031,61061,61090,61120,61149,61179,61208,61237,%
7425   61267,61296,61326,61356,61385,61415,61445,61474,61504,61533,%
7426   61563,61592,61621,61651,61680,61710,61739,61769,61799,61828,%
7427   61858,61888,61917,61947,61976,62006,62035,62064,62094,62123,%
7428   62153,62182,62212,62242,62271,62301,62331,62360,62390,62419,%
7429   62448,62478,62507,62537,62566,62596,62625,62655,62685,62715,%
7430   62744,62774,62803,62832,62862,62891,62921,62950,62980,63009,%
7431   63039,63069,63099,63128,63157,63187,63216,63246,63275,63305,%
7432   63334,63363,63393,63423,63453,63482,63512,63541,63571,63600,%
7433   63630,63659,63689,63718,63747,63777,63807,63836,63866,63895,%
7434   63925,63955,63984,64014,64043,64073,64102,64131,64161,64190,%
7435   64220,64249,64279,64309,64339,64368,64398,64427,64457,64486,%
7436   64515,64545,64574,64603,64633,64663,64692,64722,64752,64782,%
7437   64811,64841,64870,64899,64929,64958,64987,65017,65047,65076,%
7438   65106,65136,65166,65195,65225,65254,65283,65313,65342,65371,%
7439   65401,65431,65460,65490,65520}
7440 \@namedef{bbl@ca@islamic-umalqura+}{\bbl@ca@islamcuqr@x{+1}}
7441 \@namedef{bbl@ca@islamic-umalqura}{\bbl@ca@islamcuqr@x{}}
7442 \@namedef{bbl@ca@islamic-umalqura-}{\bbl@ca@islamcuqr@x{-1}}
7443 \def\bbl@ca@islamcuqr@x#1#2-#3-#4\@@#5#6#7{%
7444   \ifnum#2>2014 \ifnum#2<2038
7445     \bbl@afterfi\expandafter\@gobble
7446   \fi\fi
7447     {\bbl@error{Year~out~of~range}{The~allowed~range~is~2014-2038}}%
7448   \edef\bbl@tempd{\fp_eval:n{ % (Julian) day
7449     \bbl@ca@jd{#2}{#3}{#4} + 0.5 - 2400000 #1}}%
7450   \count@\@ne
7451   \bbl@foreach\bbl@ca@umalqura@data{%
7452     \advance\count@\@ne
7453     \ifnum##1>\bbl@tempd\else
7454       \edef\bbl@tempe{\the\count@}%
7455       \edef\bbl@tempb{##1}%
7456     \fi}%
7457   \edef\bbl@templ{\fp_eval:n{ \bbl@tempe + 16260 + 949 }}% month~lunar
7458   \edef\bbl@tempa{\fp_eval:n{ floor((\bbl@templ - 1 ) / 12) }}% annus
7459   \edef#5{\fp_eval:n{ \bbl@tempa + 1  }}%
7460   \edef#6{\fp_eval:n{ \bbl@templ - (12 * \bbl@tempa) }}%
7461   \edef#7{\fp_eval:n{ \bbl@tempd - \bbl@tempb + 1 }}}
7462 \ExplSyntaxOff
7463 \bbl@add\bbl@precalendar{%
7464   \bbl@replace\bbl@ld@calendar{-civil}{}%
7465   \bbl@replace\bbl@ld@calendar{-umalqura}{}%
7466   \bbl@replace\bbl@ld@calendar{+}{}%
7467   \bbl@replace\bbl@ld@calendar{-}{}}
```

7468 ⟨/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.

```
7469 ⟨∗ca-hebrew⟩
7470 \newcount\bbl@cntcommon
7471 \def\bbl@remainder#1#2#3{%
7472   #3 = #1                    % c = a
7473   \divide #3 by #2           % c = a/b
7474   \multiply #3 by -#2        % c = -b(a/b)
7475   \advance #3 by #1 }%        % c = a - b(a/b)
7476 \newif\ifbbl@divisible
7477 \def\bbl@checkifdivisible#1#2{%
7478   {\countdef\tmp = 0 % \tmp == \count0 - temporary variable
7479   \bbl@remainder{#1}{#2}{\tmp}%
7480   \ifnum \tmp = 0
7481       \global\bbl@divisibletrue
7482   \else
7483       \global\bbl@divisiblefalse
7484   \fi}}
7485 \newif\ifbbl@gregleap
7486 \def\bbl@ifgregleap#1{%
7487   \bbl@checkifdivisible{#1}{4}%
7488   \ifbbl@divisible
7489       \bbl@checkifdivisible{#1}{100}%
7490       \ifbbl@divisible
7491           \bbl@checkifdivisible{#1}{400}%
7492           \ifbbl@divisible
7493               \bbl@gregleaptrue
7494           \else
7495               \bbl@gregleapfalse
7496           \fi
7497       \else
7498           \bbl@gregleaptrue
7499       \fi
7500   \else
7501       \bbl@gregleapfalse
7502   \fi
7503   \ifbbl@gregleap}
7504 \def\bbl@gregdayspriormonths#1#2#3{% no month number 0
7505   {#3 = \ifcase #1 0 \or 0 \or 31 \or 59 \or 90 \or 120 \or 151 \or
7506       181 \or 212 \or 243 \or 273 \or 304 \or 334 \fi
7507   \bbl@ifgregleap{#2}%
7508       \ifnum #1 > 2        % if month after February
7509           \advance #3 by 1  % add leap day
7510       \fi
7511   \fi
7512   \global\bbl@cntcommon = #3}%
7513   #3 = \bbl@cntcommon}
7514 \def\bbl@gregdaysprioryears#1#2{%
7515   {\countdef\tmpc = 4      % \tmpc==\count4
7516   \countdef\tmpb = 2      % \tmpb==\count2
7517   \tmpb = #1              %
7518   \advance \tmpb by -1    %
7519   \tmpc = \tmpb           % \tmpc = \tmpb = year-1
7520   \multiply \tmpc by 365  % Days in prior years =
7521   #2 = \tmpc              % = 365*(year-1) ...
7522   \tmpc = \tmpb           %
7523   \divide \tmpc by 4      % \tmpc = (year-1)/4
```

207

```
7524    \advance #2 by \tmpc    % ... plus Julian leap days ...
7525    \tmpc = \tmpb          %
7526    \divide \tmpc by 100   % \tmpc = (year-1)/100
7527    \advance #2 by -\tmpc   % ... minus century years ...
7528    \tmpc = \tmpb          %
7529    \divide \tmpc by 400   % \tmpc = (year-1)/400
7530    \advance #2 by \tmpc    % ... plus 4-century years.
7531    \global\bbl@cntcommon = #2}%
7532   #2 = \bbl@cntcommon}
7533 \def\bbl@absfromgreg#1#2#3#4{%
7534   {\countdef\tmpd = 0      % \tmpd==\count0
7535    #4 = #1                 % days so far this month
7536    \bbl@gregdayspriormonths{#2}{#3}{\tmpd}%
7537    \advance #4 by \tmpd    % add days in prior months
7538    \bbl@gregdaysprioryears{#3}{\tmpd}%
7539    \advance #4 by \tmpd    % add days in prior years
7540    \global\bbl@cntcommon = #4}%
7541   #4 = \bbl@cntcommon}
7542 \newif\ifbbl@hebrleap
7543 \def\bbl@checkleaphebryear#1{%
7544   {\countdef\tmpa = 0      % \tmpa==\count0
7545    \countdef\tmpb = 1      % \tmpb==\count1
7546    \tmpa = #1
7547    \multiply \tmpa by 7
7548    \advance \tmpa by 1
7549    \bbl@remainder{\tmpa}{19}{\tmpb}%
7550    \ifnum \tmpb < 7        % \tmpb = (7*year+1)%19
7551        \global\bbl@hebrleaptrue
7552    \else
7553        \global\bbl@hebrleapfalse
7554    \fi}}
7555 \def\bbl@hebrelapsedmonths#1#2{%
7556   {\countdef\tmpa = 0      % \tmpa==\count0
7557    \countdef\tmpb = 1      % \tmpb==\count1
7558    \countdef\tmpc = 2      % \tmpc==\count2
7559    \tmpa = #1              %
7560    \advance \tmpa by -1    %
7561    #2 = \tmpa              % #2 = \tmpa = year-1
7562    \divide #2 by 19        % Number of complete Meton cycles
7563    \multiply #2 by 235     % #2 = 235*((year-1)/19)
7564    \bbl@remainder{\tmpa}{19}{\tmpb}% \tmpa = years%19-years this cycle
7565    \tmpc = \tmpb           %
7566    \multiply \tmpb by 12   %
7567    \advance #2 by \tmpb    % add regular months this cycle
7568    \multiply \tmpc by 7    %
7569    \advance \tmpc by 1     %
7570    \divide \tmpc by 19     % \tmpc = (1+7*((year-1)%19))/19 -
7571    \advance #2 by \tmpc    %  add leap months
7572    \global\bbl@cntcommon = #2}%
7573   #2 = \bbl@cntcommon}
7574 \def\bbl@hebrelapseddays#1#2{%
7575   {\countdef\tmpa = 0      % \tmpa==\count0
7576    \countdef\tmpb = 1      % \tmpb==\count1
7577    \countdef\tmpc = 2      % \tmpc==\count2
7578    \bbl@hebrelapsedmonths{#1}{#2}%
7579    \tmpa = #2              %
7580    \multiply \tmpa by 13753 %
7581    \advance \tmpa by 5604   % \tmpa=MonthsElapsed*13758 + 5604
7582    \bbl@remainder{\tmpa}{25920}{\tmpc}% \tmpc == ConjunctionParts
7583    \divide \tmpa by 25920
7584    \multiply #2 by 29
7585    \advance #2 by 1
7586    \advance #2 by \tmpa    %  #2 = 1 + MonthsElapsed*29 +
```

```
7587    \bbl@remainder{#2}{7}{\tmpa}% %  \tmpa == DayOfWeek
7588    \ifnum \tmpc < 19440
7589        \ifnum \tmpc < 9924
7590        \else                % New moon at 9 h. 204 p. or later
7591            \ifnum \tmpa = 2  % on Tuesday ...
7592                \bbl@checkleaphebryear{#1}% of a common year
7593                \ifbbl@hebrleap
7594                \else
7595                    \advance #2 by 1
7596                \fi
7597            \fi
7598        \fi
7599        \ifnum \tmpc < 16789
7600        \else                  % New moon at 15 h. 589 p. or later
7601            \ifnum \tmpa = 1   % on Monday ...
7602                \advance #1 by -1
7603                \bbl@checkleaphebryear{#1}% at the end of leap year
7604                \ifbbl@hebrleap
7605                    \advance #2 by 1
7606                \fi
7607            \fi
7608        \fi
7609    \else
7610        \advance #2 by 1       %  new moon at or after midday
7611    \fi
7612    \bbl@remainder{#2}{7}{\tmpa}% %  \tmpa == DayOfWeek
7613    \ifnum \tmpa = 0           %  if Sunday ...
7614        \advance #2 by 1
7615    \else                      %
7616        \ifnum \tmpa = 3       %  Wednesday ...
7617            \advance #2 by 1
7618        \else
7619            \ifnum \tmpa = 5   %  or Friday
7620                \advance #2 by 1
7621            \fi
7622        \fi
7623    \fi
7624    \global\bbl@cntcommon = #2}%
7625   #2 = \bbl@cntcommon}
7626 \def\bbl@daysinhebryear#1#2{%
7627  {\countdef\tmpe = 12     % \tmpe==\count12
7628    \bbl@hebrelapseddays{#1}{\tmpe}%
7629    \advance #1 by 1
7630    \bbl@hebrelapseddays{#1}{#2}%
7631    \advance #2 by -\tmpe
7632    \global\bbl@cntcommon = #2}%
7633   #2 = \bbl@cntcommon}
7634 \def\bbl@hebrdayspriormonths#1#2#3{%
7635  {\countdef\tmpf= 14     % \tmpf==\count14
7636   #3 = \ifcase #1         % Days in prior month of regular year
7637        0 \or             % no month number 0
7638        0 \or             % Tishri
7639        30 \or            % Heshvan
7640        59 \or            % Kislev
7641        89 \or            % Tebeth
7642        118 \or           % Shebat
7643        148 \or           % Adar I
7644        148 \or           % Adar II
7645        177 \or           % Nisan
7646        207 \or           % Iyar
7647        236 \or           % Sivan
7648        266 \or           % Tammuz
7649        295 \or           % Av
```

```
7650        325 \or          % Elul
7651        400              % Dummy
7652    \fi
7653    \bbl@checkleaphebryear{#2}%
7654    \ifbbl@hebrleap            % in leap year
7655        \ifnum #1 > 6          % if month after Adar I
7656            \advance #3 by 30  % add  30 days
7657        \fi
7658    \fi
7659    \bbl@daysinhebryear{#2}{\tmpf}%
7660    \ifnum #1 > 3
7661        \ifnum \tmpf = 353     %
7662            \advance #3 by -1  %
7663        \fi                    %  Short Kislev
7664        \ifnum \tmpf = 383     %
7665            \advance #3 by -1  %
7666        \fi                    %
7667    \fi
7668    \ifnum #1 > 2
7669        \ifnum \tmpf = 355     %
7670            \advance #3 by 1   %
7671        \fi                    %  Long Heshvan
7672        \ifnum \tmpf = 385     %
7673            \advance #3 by 1   %
7674        \fi                    %
7675    \fi
7676    \global\bbl@cntcommon = #3}%
7677    #3 = \bbl@cntcommon}
7678 \def\bbl@absfromhebr#1#2#3#4{%
7679    {#4 = #1
7680    \bbl@hebrdayspriormonths{#2}{#3}{#1}%
7681    \advance #4 by #1          % Add days in prior months this year
7682    \bbl@hebrelapseddays{#3}{#1}%
7683    \advance #4 by #1          % Add days in prior years
7684    \advance #4 by -1373429    % Subtract days before Gregorian
7685    \global\bbl@cntcommon = #4}%    %   01.01.0001
7686    #4 = \bbl@cntcommon}
7687 \def\bbl@hebrfromgreg#1#2#3#4#5#6{%
7688    {\countdef\tmpx= 17        % \tmpx==\count17
7689    \countdef\tmpy= 18        % \tmpy==\count18
7690    \countdef\tmpz= 19        % \tmpz==\count19
7691    #6 = #3                   %
7692    \global\advance #6 by 3761 % approximation from above
7693    \bbl@absfromgreg{#1}{#2}{#3}{#4}%
7694    \tmpz = 1   \tmpy = 1
7695    \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
7696    \ifnum \tmpx > #4              %
7697        \global\advance #6 by -1 % Hyear = Gyear + 3760
7698        \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
7699    \fi                          %
7700    \advance #4 by -\tmpx     % Days in this year
7701    \advance #4 by 1          %
7702    #5 = #4                   %
7703    \divide #5 by 30          % Approximation for month from below
7704    \loop                     % Search for month
7705        \bbl@hebrdayspriormonths{#5}{#6}{\tmpx}%
7706        \ifnum \tmpx < #4
7707            \advance #5 by 1
7708            \tmpy = \tmpx
7709    \repeat
7710    \global\advance #5 by -1
7711    \global\advance #4 by -\tmpy}}
7712 \newcount\bbl@hebrday \newcount\bbl@hebrmonth \newcount\bbl@hebryear
```

```
7713 \newcount\bbl@gregday \newcount\bbl@gregmonth \newcount\bbl@gregyear
7714 %
7715 \def\bbl@ca@hebrew#1-#2-#3\@@#4#5#6{%
7716   \bbl@gregday=#3 \bbl@gregmonth=#2 \bbl@gregyear=#1
7717   \bbl@hebrfromgreg
7718     {\bbl@gregday}{\bbl@gregmonth}{\bbl@gregyear}%
7719     {\bbl@hebrday}{\bbl@hebrmonth}{\bbl@hebryear}%
7720   \edef#4{\the\bbl@hebryear}%
7721   \edef#5{\the\bbl@hebrmonth}%
7722   \edef#6{\the\bbl@hebrday}}
7723 ⟨/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).

```
7724 ⟨*ca-persian⟩
7725 \ExplSyntaxOn
7726 \def\bbl@fpmod#1#2{(#1-#2*floor(#1/#2))}
7727 \def\bbl@cs@gregleap#1{%
7728   (\bbl@fpmod{#1}{4} == 0) &&
7729     (!((\bbl@fpmod{#1}{100} == 0) && (\bbl@fpmod{#1}{400} != 0)))}
7730 \def\bbl@ca@jd#1#2#3{% year, month, day
7731   \fp_eval:n{ 1721424.5   + (365 * (#1 - 1)) +
7732     floor((#1 - 1) / 4)   + (-floor((#1 - 1) / 100)) +
7733     floor((#1 - 1) / 400) + floor((((367 * #2) - 362) / 12) +
7734     ((#2 <= 2) ? 0 : (\bbl@cs@gregleap{#1} ? -1 : -2)) + #3) }}
7735 \def\bbl@ca@firstjal@xx{2012,2016,2020,2024,2028,2029,% March 20
7736   2032,2033,2036,2037,2040,2041,2044,2045,2048,2049}
7737 \def\bbl@ca@persian#1-#2-#3\@@#4#5#6{%
7738   \edef\bbl@tempa{#1}%  20XX-03-\bbl@tempe = 1 farvardin:
7739   \ifnum\bbl@tempa>2012 \ifnum\bbl@tempa<2051
7740     \bbl@afterfi\expandafter\@gobble
7741   \fi\fi
7742     {\bbl@error{Year~out~of~range}{The~allowed~range~is~2013-2050}}%
7743   \bbl@xin@{\bbl@tempa}{\bbl@ca@firstjal@xx}%
7744   \ifin@\def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
7745   \edef\bbl@tempc{\fp_eval:n{\bbl@ca@jd{\bbl@tempa}{#2}{#3}+.5}}% current
7746   \edef\bbl@tempb{\fp_eval:n{\bbl@ca@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}% begin
7747   \ifnum\bbl@tempc<\bbl@tempb
7748     \edef\bbl@tempa{\fp_eval:n{\bbl@tempa-1}}% go back 1 year and redo
7749     \bbl@xin@{\bbl@tempa}{\bbl@ca@firstjal@xx}%
7750     \ifin@\def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
7751     \edef\bbl@tempb{\fp_eval:n{\bbl@ca@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}%
7752   \fi
7753   \edef#4{\fp_eval:n{\bbl@tempa-621}}% set Jalali year
7754   \edef#6{\fp_eval:n{\bbl@tempc-\bbl@tempb+1}}% days from 1 farvardin
7755   \edef#5{\fp_eval:n{% set Jalali month
7756     (#6 <= 186) ? ceil(#6 / 31) : ceil((#6 - 6) / 30)}}
7757   \edef#6{\fp_eval:n{% set Jalali day
7758     (#6 - ((#5 <= 7) ? ((#5 - 1) * 31) : (((#5 - 1) * 30) + 6)))}}}
7759 \ExplSyntaxOff
7760 ⟨/ca-persian⟩
```

## 18   Buddhist

That's very simple.

```
7761 ⟨*ca-buddhist⟩
```

```
7762 \def\bbl@ca@buddhist#1-#2-#3\@@#4#5#6{%
7763   \edef#4{\number\numexpr#1+543\relax}%
7764   \edef#5{#2}%
7765   \edef#6{#3}}
7766 ⟨/ca-buddhist⟩
```

# 19  Support for Plain TeX (`plain.def`)

## 19.1  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.

```
7767 ⟨*bplain | blplain⟩
7768 \catcode`\{=1 % left brace is begin-group character
7769 \catcode`\}=2 % right brace is end-group character
7770 \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).

```
7771 \openin 0 hyphen.cfg
7772 \ifeof0
7773 \else
7774   \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.

```
7775   \def\input #1 {%
7776     \let\input\a
7777     \a hyphen.cfg
7778     \let\a\undefined
7779   }
7780 \fi
7781 ⟨/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.

```
7782 ⟨bplain⟩\a plain.tex
7783 ⟨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.

```
7784 ⟨bplain⟩\def\fmtname{babel-plain}
7785 ⟨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.

## 19.2 Emulating some LaTeX features

The file babel.def expects some definitions made in the LaTeX 2ε 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 \babeloptionstrings and \babeloptionmath are provided, which can be defined before loading babel. \BabelModifiers can be set too (but not sure it works).

```
7786 ⟨⟨∗Emulate LaTeX⟩⟩ ≡
7787 \def\@empty{}
7788 \def\loadlocalcfg#1{%
7789   \openin0#1.cfg
7790   \ifeof0
7791     \closein0
7792   \else
7793     \closein0
7794     {\immediate\write16{********************************}%
7795     \immediate\write16{* Local config file #1.cfg used}%
7796     \immediate\write16{*}%
7797     }
7798     \input #1.cfg\relax
7799   \fi
7800   \@endofldf}
```

## 19.3 General tools

A number of LaTeX macro's that are needed later on.

```
7801 \long\def\@firstofone#1{#1}
7802 \long\def\@firstoftwo#1#2{#1}
7803 \long\def\@secondoftwo#1#2{#2}
7804 \def\@nnil{\@nil}
7805 \def\@gobbletwo#1#2{}
7806 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
7807 \def\@star@or@long#1{%
7808   \@ifstar
7809   {\let\l@ngrel@x\relax#1}%
7810   {\let\l@ngrel@x\long#1}}
7811 \let\l@ngrel@x\relax
7812 \def\@car#1#2\@nil{#1}
7813 \def\@cdr#1#2\@nil{#2}
7814 \let\@typeset@protect\relax
7815 \let\protected@edef\edef
7816 \long\def\@gobble#1{}
7817 \edef\@backslashchar{\expandafter\@gobble\string\\}
7818 \def\strip@prefix#1>{}
7819 \def\g@addto@macro#1#2{{%
7820     \toks@\expandafter{#1#2}%
7821     \xdef#1{\the\toks@}}}
7822 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
7823 \def\@nameuse#1{\csname #1\endcsname}
7824 \def\@ifundefined#1{%
7825   \expandafter\ifx\csname#1\endcsname\relax
7826     \expandafter\@firstoftwo
7827   \else
7828     \expandafter\@secondoftwo
7829   \fi}
7830 \def\@expandtwoargs#1#2#3{%
7831   \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
7832 \def\zap@space#1 #2{%
7833   #1%
7834   \ifx#2\@empty\else\expandafter\zap@space\fi
7835   #2}
7836 \let\bbl@trace\@gobble
7837 \def\bbl@error#1#2{%
```

213

```
7838    \begingroup
7839      \newlinechar=`\^^J
7840      \def\\{^^J(babel) }%
7841      \errhelp{#2}\errmessage{\\#1}%
7842    \endgroup}
7843 \def\bbl@warning#1{%
7844    \begingroup
7845      \newlinechar=`\^^J
7846      \def\\{^^J(babel) }%
7847      \message{\\#1}%
7848    \endgroup}
7849 \let\bbl@infowarn\bbl@warning
7850 \def\bbl@info#1{%
7851    \begingroup
7852      \newlinechar=`\^^J
7853      \def\\{^^J}%
7854      \wlog{#1}%
7855    \endgroup}
```

LaTeX 2$_\varepsilon$ has the command \@onlypreamble which adds commands to a list of commands that are no longer needed after \begin{document}.

```
7856 \ifx\@preamblecmds\@undefined
7857    \def\@preamblecmds{}
7858 \fi
7859 \def\@onlypreamble#1{%
7860    \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
7861      \@preamblecmds\do#1}}
7862 \@onlypreamble\@onlypreamble
```

Mimick LaTeX's \AtBeginDocument; for this to work the user needs to add \begindocument to his file.

```
7863 \def\begindocument{%
7864    \@begindocumenthook
7865    \global\let\@begindocumenthook\@undefined
7866    \def\do##1{\global\let##1\@undefined}%
7867    \@preamblecmds
7868    \global\let\do\noexpand}
7869 \ifx\@begindocumenthook\@undefined
7870    \def\@begindocumenthook{}
7871 \fi
7872 \@onlypreamble\@begindocumenthook
7873 \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.

```
7874 \def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
7875 \@onlypreamble\AtEndOfPackage
7876 \def\@endofldf{}
7877 \@onlypreamble\@endofldf
7878 \let\bbl@afterlang\@empty
7879 \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.

```
7880 \catcode`\&=\z@
7881 \ifx&if@filesw\@undefined
7882    \expandafter\let\csname if@filesw\expandafter\endcsname
7883      \csname iffalse\endcsname
7884 \fi
7885 \catcode`\&=4
```

Mimick LaTeX's commands to define control sequences.

```
7886 \def\newcommand{\@star@or@long\new@command}
```

```
7887 \def\new@command#1{%
7888   \@testopt{\@newcommand#1}0}
7889 \def\@newcommand#1[#2]{%
7890   \@ifnextchar [{\@xargdef#1[#2]}%
7891                 {\@argdef#1[#2]}}
7892 \long\def\@argdef#1[#2]#3{%
7893   \@yargdef#1\@ne{#2}{#3}}
7894 \long\def\@xargdef#1[#2][#3]#4{%
7895   \expandafter\def\expandafter#1\expandafter{%
7896     \expandafter\@protected@testopt\expandafter #1%
7897     \csname\string#1\expandafter\endcsname{#3}}%
7898   \expandafter\@yargdef \csname\string#1\endcsname
7899   \tw@{#2}{#4}}
7900 \long\def\@yargdef#1#2#3{%
7901   \@tempcnta#3\relax
7902   \advance \@tempcnta \@ne
7903   \let\@hash@\relax
7904   \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
7905   \@tempcntb #2%
7906   \@whilenum\@tempcntb <\@tempcnta
7907   \do{%
7908     \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
7909     \advance\@tempcntb \@ne}%
7910   \let\@hash@##%
7911   \l@ngrel@x\expandafter\def\expandafter#1\reserved@a}
7912 \def\providecommand{\@star@or@long\provide@command}
7913 \def\provide@command#1{%
7914   \begingroup
7915     \escapechar\m@ne\xdef\@gtempa{{\string#1}}%
7916   \endgroup
7917   \expandafter\@ifundefined\@gtempa
7918     {\def\reserved@a{\new@command#1}}%
7919     {\let\reserved@a\relax
7920      \def\reserved@a{\new@command\reserved@a}}%
7921   \reserved@a}%

7922 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
7923 \def\declare@robustcommand#1{%
7924   \edef\reserved@a{\string#1}%
7925   \def\reserved@b{#1}%
7926   \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
7927   \edef#1{%
7928     \ifx\reserved@a\reserved@b
7929       \noexpand\x@protect
7930       \noexpand#1%
7931     \fi
7932     \noexpand\protect
7933     \expandafter\noexpand\csname
7934       \expandafter\@gobble\string#1 \endcsname
7935   }%
7936   \expandafter\new@command\csname
7937     \expandafter\@gobble\string#1 \endcsname
7938 }
7939 \def\x@protect#1{%
7940   \ifx\protect\@typeset@protect\else
7941     \@x@protect#1%
7942   \fi
7943 }
7944 \catcode`\&=\z@  % Trick to hide conditionals
7945   \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.

```
7946  \def\bbl@tempa{\csname newif\endcsname&ifin@}
7947 \catcode`\&=4
7948 \ifx\in@\@undefined
7949   \def\in@#1#2{%
7950     \def\in@@##1#1##2##3\in@@{%
7951       \ifx\in@##2\in@false\else\in@true\fi}%
7952     \in@@#2#1\in@\in@@}
7953 \else
7954   \let\bbl@tempa\@empty
7955 \fi
7956 \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).

```
7957 \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.

```
7958 \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.

```
7959 \ifx\@tempcnta\@undefined
7960   \csname newcount\endcsname\@tempcnta\relax
7961 \fi
7962 \ifx\@tempcntb\@undefined
7963   \csname newcount\endcsname\@tempcntb\relax
7964 \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).

```
7965 \ifx\bye\@undefined
7966   \advance\count10 by -2\relax
7967 \fi
7968 \ifx\@ifnextchar\@undefined
7969   \def\@ifnextchar#1#2#3{%
7970     \let\reserved@d=#1%
7971     \def\reserved@a{#2}\def\reserved@b{#3}%
7972     \futurelet\@let@token\@ifnch}
7973   \def\@ifnch{%
7974     \ifx\@let@token\@sptoken
7975       \let\reserved@c\@xifnch
7976     \else
7977       \ifx\@let@token\reserved@d
7978         \let\reserved@c\reserved@a
7979       \else
7980         \let\reserved@c\reserved@b
7981       \fi
7982     \fi
7983     \reserved@c}
7984   \def\:{\let\@sptoken= } \:  % this makes \@sptoken a space token
7985   \def\:{\@xifnch} \expandafter\def\: {\futurelet\@let@token\@ifnch}
7986 \fi
7987 \def\@testopt#1#2{%
7988   \@ifnextchar[{#1}{#1[#2]}}
7989 \def\@protected@testopt#1{%
7990   \ifx\protect\@typeset@protect
7991     \expandafter\@testopt
7992   \else
7993     \@x@protect#1%
```

```
7994    \fi}
7995 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
7996       #2\relax}\fi}
7997 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
7998          \else\expandafter\@gobble\fi{#1}}
```

## 19.4   Encoding related macros

Code from `ltoutenc.dtx`, adapted for use in the plain TeX environment.

```
7999 \def\DeclareTextCommand{%
8000    \@dec@text@cmd\providecommand
8001 }
8002 \def\ProvideTextCommand{%
8003    \@dec@text@cmd\providecommand
8004 }
8005 \def\DeclareTextSymbol#1#2#3{%
8006    \@dec@text@cmd\chardef#1{#2}#3\relax
8007 }
8008 \def\@dec@text@cmd#1#2#3{%
8009    \expandafter\def\expandafter#2%
8010       \expandafter{%
8011          \csname#3-cmd\expandafter\endcsname
8012          \expandafter#2%
8013          \csname#3\string#2\endcsname
8014       }%
8015 %    \let\@ifdefinable\@rc@ifdefinable
8016    \expandafter#1\csname#3\string#2\endcsname
8017 }
8018 \def\@current@cmd#1{%
8019    \ifx\protect\@typeset@protect\else
8020       \noexpand#1\expandafter\@gobble
8021    \fi
8022 }
8023 \def\@changed@cmd#1#2{%
8024    \ifx\protect\@typeset@protect
8025       \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
8026          \expandafter\ifx\csname ?\string#1\endcsname\relax
8027             \expandafter\def\csname ?\string#1\endcsname{%
8028                \@changed@x@err{#1}%
8029             }%
8030          \fi
8031          \global\expandafter\let
8032            \csname\cf@encoding \string#1\expandafter\endcsname
8033            \csname ?\string#1\endcsname
8034       \fi
8035       \csname\cf@encoding\string#1%
8036          \expandafter\endcsname
8037    \else
8038       \noexpand#1%
8039    \fi
8040 }
8041 \def\@changed@x@err#1{%
8042    \errhelp{Your command will be ignored, type <return> to proceed}%
8043    \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
8044 \def\DeclareTextCommandDefault#1{%
8045    \DeclareTextCommand#1?%
8046 }
8047 \def\ProvideTextCommandDefault#1{%
8048    \ProvideTextCommand#1?%
8049 }
8050 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
8051 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
8052 \def\DeclareTextAccent#1#2#3{%
```

```latex
8053    \DeclareTextCommand#1{#2}[1]{\accent#3 ##1}
8054 }
8055 \def\DeclareTextCompositeCommand#1#2#3#4{%
8056    \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
8057    \edef\reserved@b{\string##1}%
8058    \edef\reserved@c{%
8059       \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
8060    \ifx\reserved@b\reserved@c
8061       \expandafter\expandafter\expandafter\ifx
8062          \expandafter\@car\reserved@a\relax\relax\@nil
8063          \@text@composite
8064       \else
8065          \edef\reserved@b##1{%
8066             \def\expandafter\noexpand
8067                \csname#2\string#1\endcsname####1{%
8068                \noexpand\@text@composite
8069                   \expandafter\noexpand\csname#2\string#1\endcsname
8070                   ####1\noexpand\@empty\noexpand\@text@composite
8071                   {##1}%
8072             }%
8073          }%
8074          \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
8075       \fi
8076       \expandafter\def\csname\expandafter\string\csname
8077          #2\endcsname\string#1-\string#3\endcsname{#4}
8078    \else
8079       \errhelp{Your command will be ignored, type <return> to proceed}%
8080       \errmessage{\string\DeclareTextCompositeCommand\space used on
8081          inappropriate command \protect#1}
8082    \fi
8083 }
8084 \def\@text@composite#1#2#3\@text@composite{%
8085    \expandafter\@text@composite@x
8086       \csname\string#1-\string#2\endcsname
8087 }
8088 \def\@text@composite@x#1#2{%
8089    \ifx#1\relax
8090       #2%
8091    \else
8092       #1%
8093    \fi
8094 }
8095 %
8096 \def\@strip@args#1:#2-#3\@strip@args{#2}
8097 \def\DeclareTextComposite#1#2#3#4{%
8098    \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
8099    \bgroup
8100       \lccode`\@=#4%
8101       \lowercase{%
8102    \egroup
8103       \reserved@a @%
8104    }%
8105 }
8106 %
8107 \def\UseTextSymbol#1#2{#2}
8108 \def\UseTextAccent#1#2#3{}
8109 \def\@use@text@encoding#1{}
8110 \def\DeclareTextSymbolDefault#1#2{%
8111    \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
8112 }
8113 \def\DeclareTextAccentDefault#1#2{%
8114    \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
8115 }
```

```
8116 \def\cf@encoding{OT1}
```

Currently we only use the LaTeX 2ε method for accents for those that are known to be made active in *some* language definition file.

```
8117 \DeclareTextAccent{\"}{OT1}{127}
8118 \DeclareTextAccent{\'}{OT1}{19}
8119 \DeclareTextAccent{\^}{OT1}{94}
8120 \DeclareTextAccent{\`}{OT1}{18}
8121 \DeclareTextAccent{\~}{OT1}{126}
```

The following control sequences are used in babel.def but are not defined for PLAIN TeX.

```
8122 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
8123 \DeclareTextSymbol{\textquotedblright}{OT1}{`\"}
8124 \DeclareTextSymbol{\textquoteleft}{OT1}{`\`}
8125 \DeclareTextSymbol{\textquoteright}{OT1}{`\'}
8126 \DeclareTextSymbol{\i}{OT1}{16}
8127 \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.

```
8128 \ifx\scriptsize\@undefined
8129   \let\scriptsize\sevenrm
8130 \fi
```

And a few more "dummy" definitions.

```
8131 \def\languagename{english}%
8132 \let\bbl@opt@shorthands\@nnil
8133 \def\bbl@ifshorthand#1#2#3{#2}%
8134 \let\bbl@language@opts\@empty
8135 \ifx\babeloptionstrings\@undefined
8136   \let\bbl@opt@strings\@nnil
8137 \else
8138   \let\bbl@opt@strings\babeloptionstrings
8139 \fi
8140 \def\BabelStringsDefault{generic}
8141 \def\bbl@tempa{normal}
8142 \ifx\babeloptionmath\bbl@tempa
8143   \def\bbl@mathnormal{\noexpand\textormath}
8144 \fi
8145 \def\AfterBabelLanguage#1#2{}
8146 \ifx\BabelModifiers\@undefined\let\BabelModifiers\relax\fi
8147 \let\bbl@afterlang\relax
8148 \def\bbl@opt@safe{BR}
8149 \ifx\@uclclist\@undefined\let\@uclclist\@empty\fi
8150 \ifx\bbl@trace\@undefined\def\bbl@trace#1{}\fi
8151 \expandafter\newif\csname ifbbl@single\endcsname
8152 \chardef\bbl@bidimode\z@
8153 ⟨⟨/Emulate LaTeX⟩⟩
```

A proxy file:

```
8154 ⟨*plain⟩
8155 \input babel.def
8156 ⟨/plain⟩
```

# 20  Acknowledgements

# References

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

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

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

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

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

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

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

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

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

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

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

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