Babel

Javier Bezos
Current maintainer

Johannes L. Braams
Original author

Localization and internationalization

Unicode

T$_E$X

pdfT$_E$X

LuaT$_E$X

XeT$_E$X

Contents

Troubleshoooting

3

Part I
User guide

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

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

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

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

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

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

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

1 The user interface

1.1 Monolingual documents

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

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

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

```
PDFTEX
    \documentclass{article}

    \usepackage[T1]{fontenc}
```

```
\usepackage[french]{babel}

\begin{document}

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

\end{document}
```

Now consider something like:

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

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

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

```
\documentclass[russian]{article}

\usepackage{babel}

\babelfont{rm}{DejaVu Serif}

\begin{document}

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

\end{document}
```

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

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

Or the more explanatory:

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

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

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

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

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

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

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

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

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

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

1.2 Multilingual documents

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

EXAMPLE In LaTeX, the preamble of the document:

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

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

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

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

Examples of cases where main is useful are the following.

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

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

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

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

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

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

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

PDFTEX

```
\documentclass{article}

\usepackage[T1]{fontenc}

\usepackage[english,french]{babel}

\begin{document}

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

\selectlanguage{english}

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

\end{document}
```

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

LUATEX/XETEX

```
\documentclass{article}

\usepackage[vietnamese,danish]{babel}

\begin{document}

\prefacename, \alsoname, \today.

\selectlanguage{vietnamese}

\prefacename, \alsoname, \today.

\end{document}
```

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

1.3 Mostly monolingual documents

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

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

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

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

\babelfont[russian]{rm}{FreeSerif}

\begin{document}

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

\end{document}
```

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

1.4   Modifiers

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

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

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

1.5   Troubleshooting

- Loading directly sty files in LaTeX (ie, \usepackage{⟨*language*⟩}) is deprecated and you will get the error:[2]

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

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

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

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

---

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

1.6 Plain

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

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

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

1.7 Basic language selectors

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

\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).

\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 უძიდრესია 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]{1ຕ 1ຩ 1ອ 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]	asa	Asu
agq	Aghem	ast	Asturian[ul]
ak	Akan	az-Cyrl	Azerbaijani
am	Amharic[ul]	az-Latn	Azerbaijani
ar	Arabic[ul]	az	Azerbaijani[ul]
ar-DZ	Arabic[ul]	bas	Basaa
ar-EG	Arabic[ul]	be	Belarusian[ul]
ar-IQ	Arabic[ul]	bem	Bemba
ar-JO	Arabic[ul]	bez	Bena
ar-LB	Arabic[ul]	bg	Bulgarian[ul]
ar-MA	Arabic[ul]	bm	Bambara
ar-PS	Arabic[ul]	bn	Bangla[ul]
ar-SA	Arabic[ul]	bo	Tibetan[u]
ar-SY	Arabic[ul]	brx	Bodo
ar-TN	Arabic[ul]	bs-Cyrl	Bosnian
as	Assamese	bs-Latn	Bosnian[ul]

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

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

chechen
cherokee
chiga
chinese-hans-hk
chinese-hans-mo
chinese-hans-sg
chinese-hans
chinese-hant-hk
chinese-hant-mo
chinese-hant
chinese-simplified-hongkongsarchina
chinese-simplified-macausarchina
chinese-simplified-singapore
chinese-simplified
chinese-traditional-hongkongsarchina
chinese-traditional-macausarchina
chinese-traditional
chinese
churchslavic
churchslavic-cyrs
churchslavic-oldcyrillic[12]
churchsslavic-glag
churchsslavic-glagolitic
colognian
cornish
croatian
czech
danish
duala
dutch
dzongkha
embu
english-au
english-australia
english-ca
english-canada
english-gb
english-newzealand
english-nz
english-unitedkingdom
english-unitedstates
english-us
english
esperanto
estonian
ewe
ewondo
faroese

---

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

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

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

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

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

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

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

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

1.14 Selecting fonts

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

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

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

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

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

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

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

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

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

LUATEX/XETEX

```
\documentclass{article}

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

\babelprovide[import]{hebrew}

\babelfont{rm}{FreeSerif}

\begin{document}

Svenska \foreignlanguage{hebrew}{עִבְרִית} svenska.

\end{document}
```

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

---

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

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

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

EXAMPLE Here is how to do it:

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

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

NOTE You may load fontspec explicitly. For example:

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

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

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

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

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

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

TROUBLESHOOTING *Package 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.

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

1.15 Modifying a language

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

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

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

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

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

NOTE There are a few alternative methods:

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

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

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

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

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

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

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

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

  This redefinition is immediate.

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

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

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

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

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

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

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

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

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

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

1.16 Creating a language

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

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

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

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

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

EXAMPLE If you need a language named `arhinish`:

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

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

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

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

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

import= ⟨*language-tag*⟩

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

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

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

New 3.23 It may be used without a value. In such a case, the `ini` file set in the corresponding `babel-<language>.tex` (where `<language>` is the last argument in `\babelprovide`) is imported. See the list of recognized languages above. So, the previous example can be written:

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

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

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

captions= ⟨*language-tag*⟩

Loads only the strings. For example:

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

hyphenrules= ⟨*language-list*⟩

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

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

If none of the listed hyphenrules exist, the default behavior applies. Note in this example we set `chavacano` as first option – without it, it would select `spanish` even if `chavacano` exists.

A special value is +, which allocates a new language (in the TeX sense). It only makes sense as the last value (or the only one; the subsequent ones are silently ignored). It is mostly useful with luatex, because you can add some patterns with `\babelpatterns`, as for example:

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

In other engines it just suppresses hyphenation (because the pattern list is empty). New 3.58 Another special value is `unhyphenated`, which activates a line breking mode that allows spaces to be stretched to arbitrary amounts.

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

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

script= ⟨*script-name*⟩

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

language= ⟨*language-name*⟩

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

alph= ⟨*counter-name*⟩

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

Alph= ⟨*counter-name*⟩

Same for `\Alph`.

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

onchar= `ids` | `fonts`

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>
</dl>

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

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

intrapenalty= ⟨*penalty*⟩

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

transforms= ⟨*transform-list*⟩

See section 1.21.

justification= kashida | elongated | unhyphenated

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

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

mapfont= direction

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

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

1.17 Digits and counters

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

```
\babelprovide[import]{telugu}
  % Or also, if you want:
  % \babelprovide[import, maparabic]{telugu}
\babelfont{rm}{Gautami} % With luatex, better with Harfbuzz
\begin{document}
\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`
Bangla `alphabetic`
Central Kurdish `alphabetic`
Chinese `cjk-earthly-branch`, `cjk-heavenly-stem`, `circled.ideograph`,
   `parenthesized.ideograph`, `fullwidth.lower.alpha`, `fullwidth.upper.alpha`
Church Slavic (Glagolitic) `letters`
Coptic `epact`, `lower.letters`
French `date.day` (mainly for internal use).
Georgian `letters`
Greek `lower.modern`, `upper.modern`, `lower.ancient`, `upper.ancient` (all with keraia)
Hebrew `letters` (neither geresh nor gershayim yet)
Hindi `alphabetic`
Italian `lower.legal`, `upper.legal`
Japanese `hiragana`, `hiragana.iroha`, `katakana`, `katakana.iroha`, `circled.katakana`,
   `informal`, `formal`, `cjk-earthly-branch`, `cjk-heavenly-stem`, `circled.ideograph`,
   `parenthesized.ideograph`, `fullwidth.lower.alpha`, `fullwidth.upper.alpha`

Khmer consonant
Korean consonant, syllabe, hanja.informal, hanja.formal, hangul.formal,
cjk-earthly-branch, cjk-heavenly-stem, circled.ideograph,
parenthesized.ideograph, fullwidth.lower.alpha, fullwidth.upper.alpha
Marathi alphabetic
Persian abjad, alphabetic
Russian lower, lower.full, upper, upper.full
Syriac letters
Tamil ancient
Thai alphabetic
Ukrainian lower , lower.full, upper , upper.full

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

1.18 Dates

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

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

By default the calendar is the Gregorian, but an ini file may define strings for other
calendars (currently ar, ar-*, he, fa, hi). In the latter case, the three arguments are the
year, the month, and the day in those in the corresponding calendar. They are *not* the
Gregorian data to be converted (which means, say, 13 is a valid month number with
calendar=hebrew and calendar=coptic). However, with the option convert it's
converted (using internally the following command).
Even with a certain calendar there may be variants. In Kurmanji the default variant prints
something like *30. Çileya Pêşîn 2019*, but with variant=izafa it prints *31'ê Çileya Pêşînê
2019*.

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

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

1.19 Accessing language info

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

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

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

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

\localeinfo ***{⟨*field*⟩}**

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

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

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

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

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

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

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

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

\localeid Each language in the babel sense has its own unique numeric identifier, which can be retrieved with \localeid.
The \localeid is not the same as the \language identifier, which refers to a set of hyphenation patters (which, in turn, is just a component of the line breaking algorithm described in the next section). The data about preloaded patterns are store in an internal macro named \bbl@languages (see the code for further details), but note several locales may share a single \language, so they are separated concepts. In luatex, the \localeid is saved in each node (when it makes sense) as an attribute, too.

`\LocaleForEach` {⟨*code*⟩}

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

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

1.20 Hyphenation and line breaking

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

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

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

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

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

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

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

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

With all of them, hyphenation in the rest of the word is enabled. If you don't want to enable it, there is a starred counterpart: `\babelhyphen*{soft}` (which in most cases is equivalent to the original `\-`), `\babelhyphen*{hard}`, etc.
Note `hard` is also good for isolated prefixes (eg, *anti-*) and `nobreak` for isolated suffixes (eg, *-ism*), but in both cases `\babelhyphen*{nobreak}` is usually better.
There are also some differences with LaTeX: (1) the character used is that set for the current font, while in LaTeX it is hardwired to `-` (a typical value); (2) the hyphen to be used in fonts with a negative `\hyphenchar` is `-`, like in LaTeX, but it can be changed to another value by redefining `\babelnullhyphen`; (3) a break after the hyphen is forbidden if preceded by a glue >0 pt (at the beginning of a word, provided it is not immediately preceded by, say, a parenthesis).

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

New 3.9a Sets hyphenation exceptions for the languages given or, without the optional argument, for *all* languages (eg, proper nouns or common loan words, and of course monolingual documents). Multiple declarations work much like `\hyphenation` (last wins), but language exceptions take precedence over global ones.

It can be used only in the preamble, and exceptions are set when the language is first selected, thus taking into account changes of `\lccode`'s done in `\extras`⟨*lang*⟩ as well as the language-specific encoding (not set in the preamble by default). Multiple `\babelhyphenation`'s are allowed. For example:

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

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

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

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

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

The environment hyphenrules can be used to select *only* the hyphenation rules to be used (it can be used as command, too). This can for instance be used to select 'nohyphenation', provided that in `language.dat` the 'language' nohyphenation is defined by loading `zerohyph.tex`. It deactivates language shorthands, too (but not user shorthands).

Except for these simple uses, hyphenrules is deprecated and `otherlanguage*` (the starred version) is preferred, because the former does not take into account possible changes in encodings of characters like, say, ' done by some languages (eg, italian, french, ukraineb).

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

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

It can be used only in the preamble, and patterns are added when the language is first selected, thus taking into account changes of `\lccode`'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. (A few may still require some fine-tuning. More to follow in future releases.)

Arabic	`transliteration.dad`	Applies the transliteration system devised by Yannis Haralambous for dad (simple and TeX-friendly). Not yet complete, but sufficient for most texts.
Croatian	`digraphs.ligatures`	Ligatures *DŽ, Dž, dž, LJ, Lj, lj, NJ, Nj, nj*. It assumes they exist. This is not the recommended way to make these transformations (the best way is with OTF features), but it can get you out of a hurry.
Czech, Polish, Portuguese, Slovak, Spanish	`hyphen.repeat`	Explicit hyphens behave like \babelhyphen {repeat}.
Czech, Polish, Slovak	`oneletter.nobreak`	Converts a space after a non-syllabic preposition or conjunction into a non-breaking space.
Finnish	`prehyphen.nobreak`	Line breaks just after hyphens prepended to words are prevented, like in "pakastekaapit ja -arkut".
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 ([ï̆ü̆]), the replacement could be {1|ï̆ü̆|íú}, which maps *ï̆* to *í*, and *ü̆* 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) بــ
        Arabia أو Aravia (بالاغريقية Αραβία) ، استخدم الرومان ثلاث
        بادئات بــ"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 العصر فصحى \textit{fuṣḥā l-'aṣr} (MSA) and
التراث فصحى \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).

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

56

\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=3.80.2871⟩⟩
2 ⟨⟨date=2022/09/25⟩⟩
```

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 ϵ-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}}}
```

62

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 extensison 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     \PackageNote{babel}{#1}%
240   \endgroup}
241 \def\bbl@info#1{%
242   \begingroup
243     \def\\{\MessageBreak}%
244     \PackageInfo{babel}{#1}%
245   \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.

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

```
255 \ifx\bbl@languages\@undefined\else
256   \begingroup
257     \catcode`\^^I=12
258     \@ifpackagewith{babel}{showlanguages}{%
259       \begingroup
```

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

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

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

67

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

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

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

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

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

```
355 \let\bbl@language@opts\@empty
356 \DeclareOption*{%
```

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

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

```
363 \ProcessOptions*

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

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=....

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

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

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

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

```
397    \edef\bbl@opt@shorthands{%
398      \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.

```
399    \bbl@ifshorthand{'}%
400      {\PassOptionsToPackage{activeacute}{babel}}{}
401    \bbl@ifshorthand{`}%
402      {\PassOptionsToPackage{activegrave}{babel}}{}
403 \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.

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

For the option safe we use a different approach – `\bbl@opt@safe` says which macros are redefined (B for bibs and R for refs). By default, both are currently set, but in a future release it will be set to none.

```
410 \ifx\bbl@opt@safe\@undefined
411   \def\bbl@opt@safe{BR}
412   % \let\bbl@opt@safe\@empty % Pending of \cite
413 \fi
```

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

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

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

```
435 ⟨/core⟩
436 ⟨*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.

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

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

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

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

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

71

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

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

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

```
558 \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.

```
559 \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.

```
560 \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
```
561 \def\bbl@push@language{%
562   \ifx\languagename\@undefined\else
563     \ifx\currentgrouplevel\@undefined
564       \xdef\bbl@language@stack{\languagename+\bbl@language@stack}%
565     \else
566       \ifnum\currentgrouplevel=\z@
567         \xdef\bbl@language@stack{\languagename+}%
568       \else
569         \xdef\bbl@language@stack{\languagename+\bbl@language@stack}%
570       \fi
571     \fi
572   \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.

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

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

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

The unprotected part of \selectlanguage.

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

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

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

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

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

otherlanguage (*env.*) The otherlanguage environment can be used as an alternative to using the \selectlanguage declarative command. When you are typesetting a document which mixes left-to-right and right-to-left typesetting you have to use this environment in order to let things work as you expect them to.

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

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

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

```
770 \long\def\endotherlanguage{%
```

```
771    \global\@ignoretrue\ignorespaces}
```

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

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

```
779 \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.

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

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

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

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

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

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

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

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

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

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

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

```
886  \def\set@hyphenmins#1#2{%
887    \lefthyphenmin#1\relax
888    \righthyphenmin#2\relax}
```

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

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

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

```
905 \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.

```
906 \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':

```
907 \providecommand\setlocale{%
908    \bbl@error
909       {Not yet available}%
910       {Find an armchair, sit down and wait}}
911 \let\uselocale\setlocale
912 \let\locale\setlocale
913 \let\selectlocale\setlocale
914 \let\textlocale\setlocale
915 \let\textlanguage\setlocale
916 \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 LaTeX2_ε, so we can safely use its error
handling interface. Otherwise we'll have to 'keep it simple'.
Infos are not written to the console, but on the other hand many people think warnings are errors, so
a further message type is defined: an important info which is sent to the console.

```
917 \edef\bbl@nulllanguage{\string\language=0}
918 \def\bbl@nocaption{\protect\bbl@nocaption@i}
919 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
920    \global\@namedef{#2}{\textbf{?#1?}}%
921    \@nameuse{#2}%
922    \edef\bbl@tempa{#1}%
923    \bbl@sreplace\bbl@tempa{name}{}%
924    \bbl@warning{% TODO.
925       \@backslashchar#1 not set for '\languagename'. Please,\\%
926       define it after the language has been loaded\\%
927       (typically in the preamble) with:\\%
928       \string\setlocalecaption{\languagename}{\bbl@tempa}{..}\\%
929       Feel free to contribute on github.com/latex3/babel.\\%
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.

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

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

```
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 The initialization of a new save cycle: reset the counter to zero.
\babel@beginsave

```
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   The macro \babel@save⟨csname⟩ saves the current meaning of the control sequence ⟨csname⟩ to
\babel@savevariable \originalTeX[32]. To do this, we let the current meaning to a temporary control sequence, the restore
                  commands are appended to \originalTeX and the counter is incremented. The macro
                  \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   Some languages need to have \frenchspacing in effect. Others don't want that. The command
\bbl@nonfrenchspacing \bbl@frenchspacing switches it on when it isn't already in effect and \bbl@nonfrenchspacing
                       switches it off if necessary. A more refined way to switch the catcodes is done with ini files. Here an
                       auxiliary macro is defined, but the main part is in \babelprovide. This new method should be
                       ideally the default one.

```
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}%
```

---

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

```
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
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}
```

---

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

\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}
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{%
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 % A temporary hack:
1772 \ifx\BabelCaseHack\@undefined
1773 \AtBeginDocument{%
1774   \bbl@exp{%
1775     \\\in@{\string\@uclclist}%
1776         {\expandafter\meaning\csname MakeUppercase \endcsname}}%
1777   \ifin@\else
1778     \expandafter\let\expandafter\bbl@newuc\csname MakeUppercase \endcsname
1779     \protected\@namedef{MakeUppercase }#1{{%
1780       \def\reserved@a##1##2{\let##1##2\reserved@a}%
1781       \expandafter\reserved@a\@uclclist\reserved@b{\reserved@b\@gobble}%
1782       \protected@edef\reserved@a{\bbl@newuc{#1}}\reserved@a}}%
1783     \expandafter\let\expandafter\bbl@newlc\csname MakeLowercase \endcsname
1784     \protected\@namedef{MakeLowercase }#1{{%
1785       \def\reserved@a##1##2{\let##2##1\reserved@a}%
1786       \expandafter\reserved@a\@uclclist\reserved@b{\reserved@b\@gobble}%
1787       \protected@edef\reserved@a{\bbl@newlc{#1}}\reserved@a}}%
1788   \fi}
1789 \fi
```

```
1790 ⟨⟨*More package options⟩⟩ ≡
1791 \DeclareOption{nocase}{}
1792 ⟨⟨/More package options⟩⟩
```

The following package options control the behavior of \SetString.

```
1793 ⟨⟨*More package options⟩⟩ ≡
1794 \let\bbl@opt@strings\@nnil % accept strings=value
1795 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
1796 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
1797 \def\BabelStringsDefault{generic}
1798 ⟨⟨/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.

```
1799 \@onlypreamble\StartBabelCommands
1800 \def\StartBabelCommands{%
1801   \begingroup
1802   \bbl@recatcode{11}%
1803   ⟨⟨Macros local to BabelCommands⟩⟩
1804   \def\bbl@provstring##1##2{%
1805     \providecommand##1{##2}%
1806     \bbl@toglobal##1}%
1807   \global\let\bbl@scafter\@empty
1808   \let\StartBabelCommands\bbl@startcmds
1809   \ifx\BabelLanguages\relax
1810       \let\BabelLanguages\CurrentOption
1811   \fi
1812   \begingroup
1813   \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
1814   \StartBabelCommands}
1815 \def\bbl@startcmds{%
1816   \ifx\bbl@screset\@nnil\else
1817     \bbl@usehooks{stopcommands}{}%
1818   \fi
1819   \endgroup
1820   \begingroup
1821   \@ifstar
1822     {\ifx\bbl@opt@strings\@nnil
1823        \let\bbl@opt@strings\BabelStringsDefault
1824      \fi
1825      \bbl@startcmds@i}%
1826     \bbl@startcmds@i}
1827 \def\bbl@startcmds@i#1#2{%
1828   \edef\bbl@L{\zap@space#1 \@empty}%
1829   \edef\bbl@G{\zap@space#2 \@empty}%
1830   \bbl@startcmds@ii}
1831 \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.

```
1832 \newcommand\bbl@startcmds@ii[1][\@empty]{%
1833   \let\SetString\@gobbletwo
1834   \let\bbl@stringdef\@gobbletwo
1835   \let\AfterBabelCommands\@gobble
1836   \ifx\@empty#1%
1837     \def\bbl@sc@label{generic}%
1838     \def\bbl@encstring##1##2{%
1839       \ProvideTextCommandDefault##1{##2}%
```

```
1840        \bbl@toglobal##1%
1841        \expandafter\bbl@toglobal\csname\string?\string##1\endcsname}%
1842      \let\bbl@sctest\in@true
1843    \else
1844      \let\bbl@sc@charset\space % <- zapped below
1845      \let\bbl@sc@fontenc\space % <-    "        "
1846      \def\bbl@tempa##1=##2\@nil{%
1847        \bbl@csarg\edef{sc@\zap@space##1 \@empty}{##2 }}%
1848      \bbl@vforeach{label=#1}{\bbl@tempa##1\@nil}%
1849      \def\bbl@tempa##1 ##2{% space -> comma
1850        ##1%
1851        \ifx\@empty##2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa##2\fi}%
1852      \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%
1853      \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%
1854      \edef\bbl@sc@charset{\expandafter\zap@space\bbl@sc@charset\@empty}%
1855      \def\bbl@encstring##1##2{%
1856        \bbl@foreach\bbl@sc@fontenc{%
1857          \bbl@ifunset{T@####1}%
1858            {}%
1859            {\ProvideTextCommand##1{####1}{##2}%
1860             \bbl@toglobal##1%
1861             \expandafter
1862             \bbl@toglobal\csname####1\string##1\endcsname}}%
1863      \def\bbl@sctest{%
1864        \bbl@xin@{,\bbl@opt@strings,}{,\bbl@sc@label,\bbl@sc@fontenc,}}%
1865    \fi
1866    \ifx\bbl@opt@strings\@nnil        % ie, no strings key -> defaults
1867    \else\ifx\bbl@opt@strings\relax   % ie, strings=encoded
1868      \let\AfterBabelCommands\bbl@aftercmds
1869      \let\SetString\bbl@setstring
1870      \let\bbl@stringdef\bbl@encstring
1871    \else         % ie, strings=value
1872    \bbl@sctest
1873    \ifin@
1874      \let\AfterBabelCommands\bbl@aftercmds
1875      \let\SetString\bbl@setstring
1876      \let\bbl@stringdef\bbl@provstring
1877    \fi\fi\fi
1878    \bbl@scswitch
1879    \ifx\bbl@G\@empty
1880      \def\SetString##1##2{%
1881        \bbl@error{Missing group for string \string##1}%
1882          {You must assign strings to some category, typically\\%
1883           captions or extras, but you set none}}%
1884    \fi
1885    \ifx\@empty#1%
1886      \bbl@usehooks{defaultcommands}{}%
1887    \else
1888      \@expandtwoargs
1889      \bbl@usehooks{encodedcommands}{{\bbl@sc@charset}{\bbl@sc@fontenc}}%
1890    \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) .

```
1891 \def\bbl@forlang#1#2{%
1892   \bbl@for#1\bbl@L{%
1893     \bbl@xin@{,#1,}{,\BabelLanguages,}%
```

```
1894     \ifin@#2\relax\fi}}
1895 \def\bbl@scswitch{%
1896   \bbl@forlang\bbl@tempa{%
1897     \ifx\bbl@G\@empty\else
1898       \ifx\SetString\@gobbletwo\else
1899         \edef\bbl@GL{\bbl@G\bbl@tempa}%
1900         \bbl@xin@{,\bbl@GL,}{,\bbl@screset,}%
1901         \ifin@\else
1902           \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
1903           \xdef\bbl@screset{\bbl@screset,\bbl@GL}%
1904         \fi
1905       \fi
1906     \fi}}
1907 \AtEndOfPackage{%
1908   \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{}{#2}}}%
1909   \let\bbl@scswitch\relax}
1910 \@onlypreamble\EndBabelCommands
1911 \def\EndBabelCommands{%
1912   \bbl@usehooks{stopcommands}{}%
1913   \endgroup
1914   \endgroup
1915   \bbl@scafter}
1916 \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.

```
1917 \def\bbl@setstring#1#2{% eg, \prefacename{<string>}
1918   \bbl@forlang\bbl@tempa{%
1919     \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
1920     \bbl@ifunset{\bbl@LC}% eg, \germanchaptername
1921       {\bbl@exp{%
1922         \global\\\bbl@add\<\bbl@G\bbl@tempa>{\\\bbl@scset\\#1\<\bbl@LC>}}}%
1923       {}%
1924     \def\BabelString{#2}%
1925     \bbl@usehooks{stringprocess}{}%
1926     \expandafter\bbl@stringdef
1927       \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.

```
1928 \ifx\bbl@opt@strings\relax
1929   \def\bbl@scset#1#2{\def#1{\bbl@encoded#2}}
1930   \bbl@patchuclc
1931   \let\bbl@encoded\relax
1932   \def\bbl@encoded@uclc#1{%
1933     \@inmathwarn#1%
1934     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
1935       \expandafter\ifx\csname ?\string#1\endcsname\relax
1936         \TextSymbolUnavailable#1%
1937       \else
1938         \csname ?\string#1\endcsname
1939       \fi
1940     \else
1941       \csname\cf@encoding\string#1\endcsname
1942     \fi}
1943 \else
1944   \def\bbl@scset#1#2{\def#1{#2}}
1945 \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.

```
1946 ⟨⟨*Macros local to BabelCommands⟩⟩ ≡
1947 \def\SetStringLoop##1##2{%
1948     \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
1949     \count@\z@
1950     \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
1951       \advance\count@\@ne
1952       \toks@\expandafter{\bbl@tempa}%
1953       \bbl@exp{%
1954         \\\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
1955         \count@=\the\count@\relax}}}%
1956 ⟨⟨/Macros local to BabelCommands⟩⟩
```

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

```
1957 \def\bbl@aftercmds#1{%
1958   \toks@\expandafter{\bbl@scafter#1}%
1959   \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.

```
1960 ⟨⟨*Macros local to BabelCommands⟩⟩ ≡
1961   \newcommand\SetCase[3][]{%
1962     \bbl@patchuclc
1963     \bbl@forlang\bbl@tempa{%
1964       \expandafter\bbl@encstring
1965         \csname\bbl@tempa @bbl@uclc\endcsname{\bbl@tempa##1}%
1966       \expandafter\bbl@encstring
1967         \csname\bbl@tempa @bbl@uc\endcsname{##2}%
1968       \expandafter\bbl@encstring
1969         \csname\bbl@tempa @bbl@lc\endcsname{##3}}}%
1970 ⟨⟨/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.

```
1971 ⟨⟨*Macros local to BabelCommands⟩⟩ ≡
1972   \newcommand\SetHyphenMap[1]{%
1973     \bbl@forlang\bbl@tempa{%
1974       \expandafter\bbl@stringdef
1975         \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
1976 ⟨⟨/Macros local to BabelCommands⟩⟩
```

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

```
1977 \newcommand\BabelLower[2]{% one to one.
1978   \ifnum\lccode#1=#2\else
1979     \babel@savevariable{\lccode#1}%
1980     \lccode#1=#2\relax
1981   \fi}
1982 \newcommand\BabelLowerMM[4]{% many-to-many
1983   \@tempcnta=#1\relax
1984   \@tempcntb=#4\relax
1985   \def\bbl@tempa{%
1986     \ifnum\@tempcnta>#2\else
1987       \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
1988       \advance\@tempcnta#3\relax
1989       \advance\@tempcntb#3\relax
1990       \expandafter\bbl@tempa
1991     \fi}%
1992   \bbl@tempa}
```

```
1993 \newcommand\BabelLowerMO[4]{% many-to-one
1994   \@tempcnta=#1\relax
1995   \def\bbl@tempa{%
1996     \ifnum\@tempcnta>#2\else
1997       \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
1998       \advance\@tempcnta#3
1999       \expandafter\bbl@tempa
2000     \fi}%
2001   \bbl@tempa}
```

The following package options control the behavior of hyphenation mapping.

```
2002 ⟨⟨*More package options⟩⟩ ≡
2003 \DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\z@}
2004 \DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap\@ne}
2005 \DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\tw@}
2006 \DeclareOption{hyphenmap=other}{\chardef\bbl@opt@hyphenmap\thr@@}
2007 \DeclareOption{hyphenmap=other*}{\chardef\bbl@opt@hyphenmap4\relax}
2008 ⟨⟨/More package options⟩⟩
```

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

```
2009 \AtEndOfPackage{%
2010   \ifx\bbl@opt@hyphenmap\@undefined
2011     \bbl@xin@{,}{\bbl@language@opts}%
2012     \chardef\bbl@opt@hyphenmap\ifin@4\else\@ne\fi
2013   \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.

```
2014 \newcommand\setlocalecaption{%  TODO. Catch typos. What about ensure?
2015   \@ifstar\bbl@setcaption@s\bbl@setcaption@x}
2016 \def\bbl@setcaption@x#1#2#3{%  language caption-name string
2017   \bbl@trim@def\bbl@tempa{#2}%
2018   \bbl@xin@{.template}{\bbl@tempa}%
2019   \ifin@
2020     \bbl@ini@captions@template{#3}{#1}%
2021   \else
2022     \edef\bbl@tempd{%
2023       \expandafter\expandafter\expandafter
2024       \strip@prefix\expandafter\meaning\csname captions#1\endcsname}%
2025     \bbl@xin@
2026       {\expandafter\string\csname #2name\endcsname}%
2027       {\bbl@tempd}%
2028     \ifin@ % Renew caption
2029       \bbl@xin@{\string\bbl@scset}{\bbl@tempd}%
2030       \ifin@
2031         \bbl@exp{%
2032           \\\bbl@ifsamestring{\bbl@tempa}{\languagename}%
2033             {\\\bbl@scset\<#2name>\<#1#2name>}%
2034             {}}%
2035       \else % Old way converts to new way
2036         \bbl@ifunset{#1#2name}%
2037           {\bbl@exp{%
2038             \\\bbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}%
2039             \\\bbl@ifsamestring{\bbl@tempa}{\languagename}%
2040               {\def\<#2name>{\<#1#2name>}}%
2041               {}}}%
2042           {}%
2043       \fi
2044     \else
2045       \bbl@xin@{\string\bbl@scset}{\bbl@tempd}% New
2046       \ifin@ % New way
2047         \bbl@exp{%
2048           \\\bbl@add\<captions#1>{\\\bbl@scset\<#2name>\<#1#2name>}%
```

```
2049          \\\bbl@ifsamestring{\bbl@tempa}{\languagename}%
2050             {\\\bbl@scset\<#2name>\<#1#2name>}%
2051             {}}%
2052        \else  % Old way, but defined in the new way
2053          \bbl@exp{%
2054            \\\bbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}%
2055            \\\bbl@ifsamestring{\bbl@tempa}{\languagename}%
2056             {\def\<#2name>{\<#1#2name>}}%
2057             {}}%
2058        \fi%
2059      \fi
2060      \@namedef{#1#2name}{#3}%
2061      \toks@\expandafter{\bbl@captionslist}%
2062      \bbl@exp{\\\in@{\<#2name>}{\the\toks@}}%
2063      \ifin@\else
2064        \bbl@exp{\\\bbl@add\\\bbl@captionslist{\<#2name>}}%
2065        \bbl@toglobal\bbl@captionslist
2066      \fi
2067    \fi}
2068 % \def\bbl@setcaption@s#1#2#3{} % TODO. Not yet implemented (w/o 'name')
```

7.11 Macros common to a number of languages

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

```
2069 \bbl@trace{Macros related to glyphs}
2070 \def\set@low@box#1{\setbox\tw@\hbox{,}\setbox\z@\hbox{#1}%
2071    \dimen\z@\ht\z@ \advance\dimen\z@ -\ht\tw@%
2072    \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.

```
2073 \def\save@sf@q#1{\leavevmode
2074    \begingroup
2075      \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
2076    \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.

```
2077 \ProvideTextCommand{\quotedblbase}{OT1}{%
2078   \save@sf@q{\set@low@box{\textquotedblright\/}%
2079     \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.

```
2080 \ProvideTextCommandDefault{\quotedblbase}{%
2081   \UseTextSymbol{OT1}{\quotedblbase}}
```

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

```
2082 \ProvideTextCommand{\quotesinglbase}{OT1}{%
2083   \save@sf@q{\set@low@box{\textquoteright\/}%
2084     \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.

```
2085 \ProvideTextCommandDefault{\quotesinglbase}{%
2086   \UseTextSymbol{OT1}{\quotesinglbase}}
```

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

```
2087 \ProvideTextCommand{\guillemetleft}{OT1}{%
2088   \ifmmode
2089     \ll
2090   \else
2091     \save@sf@q{\nobreak
2092       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2093   \fi}
2094 \ProvideTextCommand{\guillemetright}{OT1}{%
2095   \ifmmode
2096     \gg
2097   \else
2098     \save@sf@q{\nobreak
2099       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2100   \fi}
2101 \ProvideTextCommand{\guillemotleft}{OT1}{%
2102   \ifmmode
2103     \ll
2104   \else
2105     \save@sf@q{\nobreak
2106       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2107   \fi}
2108 \ProvideTextCommand{\guillemotright}{OT1}{%
2109   \ifmmode
2110     \gg
2111   \else
2112     \save@sf@q{\nobreak
2113       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2114   \fi}
```

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

```
2115 \ProvideTextCommandDefault{\guillemetleft}{%
2116   \UseTextSymbol{OT1}{\guillemetleft}}
2117 \ProvideTextCommandDefault{\guillemetright}{%
2118   \UseTextSymbol{OT1}{\guillemetright}}
2119 \ProvideTextCommandDefault{\guillemotleft}{%
2120   \UseTextSymbol{OT1}{\guillemotleft}}
2121 \ProvideTextCommandDefault{\guillemotright}{%
2122   \UseTextSymbol{OT1}{\guillemotright}}
```

\guilsinglleft The single guillemets are not available in OT1 encoding. They are faked.
\guilsinglright
```
2123 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2124   \ifmmode
2125     <%
2126   \else
2127     \save@sf@q{\nobreak
2128       \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
2129   \fi}
2130 \ProvideTextCommand{\guilsinglright}{OT1}{%
2131   \ifmmode
2132     >%
2133   \else
2134     \save@sf@q{\nobreak
2135       \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
2136   \fi}
```

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

```
2137 \ProvideTextCommandDefault{\guilsinglleft}{%
2138   \UseTextSymbol{OT1}{\guilsinglleft}}
2139 \ProvideTextCommandDefault{\guilsinglright}{%
2140   \UseTextSymbol{OT1}{\guilsinglright}}
```

7.12.2 Letters

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

```
2141 \DeclareTextCommand{\ij}{OT1}{%
2142   i\kern-0.02em\bbl@allowhyphens j}
2143 \DeclareTextCommand{\IJ}{OT1}{%
2144   I\kern-0.02em\bbl@allowhyphens J}
2145 \DeclareTextCommand{\ij}{T1}{\char188}
2146 \DeclareTextCommand{\IJ}{T1}{\char156}
```

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

```
2147 \ProvideTextCommandDefault{\ij}{%
2148   \UseTextSymbol{OT1}{\ij}}
2149 \ProvideTextCommandDefault{\IJ}{%
2150   \UseTextSymbol{OT1}{\IJ}}
```

\dj The croatian language needs the letters \dj and \DJ; they are available in the T1 encoding, but not in
\DJ the OT1 encoding by default.
     Some code to construct these glyphs for the OT1 encoding was made available to me by Stipčević
     Mario, (stipcevic@olimp.irb.hr).

```
2151 \def\crrtic@{\hrule height0.1ex width0.3em}
2152 \def\crttic@{\hrule height0.1ex width0.33em}
2153 \def\ddj@{%
2154   \setbox0\hbox{d}\dimen@=\ht0
2155   \advance\dimen@1ex
2156   \dimen@.45\dimen@
2157   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2158   \advance\dimen@ii.5ex
2159   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2160 \def\DDJ@{%
2161   \setbox0\hbox{D}\dimen@=.55\ht0
2162   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2163   \advance\dimen@ii.15ex %              correction for the dash position
2164   \advance\dimen@ii-.15\fontdimen7\font %    correction for cmtt font
2165   \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
2166   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
2167 %
2168 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2169 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}
```

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

```
2170 \ProvideTextCommandDefault{\dj}{%
2171   \UseTextSymbol{OT1}{\dj}}
2172 \ProvideTextCommandDefault{\DJ}{%
2173   \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.

```
2174 \DeclareTextCommand{\SS}{OT1}{SS}
2175 \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
```
2176 \ProvideTextCommandDefault{\glq}{%
2177   \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}
```

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

```
2178 \ProvideTextCommand{\grq}{T1}{%
2179   \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}
2180 \ProvideTextCommand{\grq}{TU}{%
2181   \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
2182 \ProvideTextCommand{\grq}{OT1}{%
2183   \save@sf@q{\kern-.0125em
2184     \textormath{\textquoteleft}{\mbox{\textquoteleft}}%
2185     \kern.07em\relax}}
2186 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}\grq}
```

\glqq   The 'german' double quotes.

\grqq
```
2187 \ProvideTextCommandDefault{\glqq}{%
2188   \textormath{\quotedblbase}{\mbox{\quotedblbase}}}
```

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

```
2189 \ProvideTextCommand{\grqq}{T1}{%
2190   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2191 \ProvideTextCommand{\grqq}{TU}{%
2192   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2193 \ProvideTextCommand{\grqq}{OT1}{%
2194   \save@sf@q{\kern-.07em
2195     \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}%
2196     \kern.07em\relax}}
2197 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}
```

\flq   The 'french' single guillemets.

\frq
```
2198 \ProvideTextCommandDefault{\flq}{%
2199   \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
2200 \ProvideTextCommandDefault{\frq}{%
2201   \textormath{\guilsinglright}{\mbox{\guilsinglright}}}
```

\flqq   The 'french' double guillemets.

\frqq
```
2202 \ProvideTextCommandDefault{\flqq}{%
2203   \textormath{\guillemetleft}{\mbox{\guillemetleft}}}
2204 \ProvideTextCommandDefault{\frqq}{%
2205   \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).

```
2206 \def\umlauthigh{%
2207   \def\bbl@umlauta##1{\leavevmode\bgroup%
2208     \expandafter\accent\csname\f@encoding dqpos\endcsname
2209     ##1\bbl@allowhyphens\egroup}%
2210   \let\bbl@umlaute\bbl@umlauta}
2211 \def\umlautlow{%
2212   \def\bbl@umlauta{\protect\lower@umlaut}}
2213 \def\umlautelow{%
2214   \def\bbl@umlaute{\protect\lower@umlaut}}
2215 \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.

```
2216 \expandafter\ifx\csname U@D\endcsname\relax
2217   \csname newdimen\endcsname\U@D
2218 \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.

```
2219 \def\lower@umlaut#1{%
2220   \leavevmode\bgroup
2221     \U@D 1ex%
2222     {\setbox\z@\hbox{%
2223       \expandafter\char\csname\f@encoding dqpos\endcsname}%
2224       \dimen@ -.45ex\advance\dimen@\ht\z@
2225       \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2226     \expandafter\accent\csname\f@encoding dqpos\endcsname
2227     \fontdimen5\font\U@D #1%
2228   \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).

```
2229 \AtBeginDocument{%
2230   \DeclareTextCompositeCommand{\"}{OT1}{a}{\bbl@umlauta{a}}%
2231   \DeclareTextCompositeCommand{\"}{OT1}{e}{\bbl@umlaute{e}}%
2232   \DeclareTextCompositeCommand{\"}{OT1}{i}{\bbl@umlaute{\i}}%
2233   \DeclareTextCompositeCommand{\"}{OT1}{\i}{\bbl@umlaute{\i}}%
2234   \DeclareTextCompositeCommand{\"}{OT1}{o}{\bbl@umlauta{o}}%
2235   \DeclareTextCompositeCommand{\"}{OT1}{u}{\bbl@umlauta{u}}%
2236   \DeclareTextCompositeCommand{\"}{OT1}{A}{\bbl@umlauta{A}}%
2237   \DeclareTextCompositeCommand{\"}{OT1}{E}{\bbl@umlaute{E}}%
2238   \DeclareTextCompositeCommand{\"}{OT1}{I}{\bbl@umlaute{I}}%
2239   \DeclareTextCompositeCommand{\"}{OT1}{O}{\bbl@umlauta{O}}%
2240   \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.

```
2241 \ifx\l@english\@undefined
2242   \chardef\l@english\z@
2243 \fi
2244 % The following is used to cancel rules in ini files (see Amharic).
2245 \ifx\l@unhyphenated\@undefined
2246   \newlanguage\l@unhyphenated
2247 \fi
```

7.13 Layout

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

```
2248 \bbl@trace{Bidi layout}
2249 \providecommand\IfBabelLayout[3]{#3}%
2250 \newcommand\BabelPatchSection[1]{%
2251   \@ifundefined{#1}{}{%
2252     \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
2253     \@namedef{#1}{%
2254       \@ifstar{\bbl@presec@s{#1}}%
2255               {\@dblarg{\bbl@presec@x{#1}}}}}}
2256 \def\bbl@presec@x#1[#2]#3{%
2257   \bbl@exp{%
2258     \\\select@language@x{\bbl@main@language}%
2259     \\\bbl@cs{sspre@#1}%
```

```
2260      \\\bbl@cs{ss@#1}%
2261        [\\\foreignlanguage{\languagename}{\unexpanded{#2}}]%
2262        {\\\foreignlanguage{\languagename}{\unexpanded{#3}}}%
2263      \\\select@language@x{\languagename}}}
2264 \def\bbl@presec@s#1#2{%
2265   \bbl@exp{%
2266      \\\select@language@x{\bbl@main@language}%
2267      \\\bbl@cs{sspre@#1}%
2268      \\\bbl@cs{ss@#1}*%
2269        {\\\foreignlanguage{\languagename}{\unexpanded{#2}}}%
2270      \\\select@language@x{\languagename}}}
2271 \IfBabelLayout{sectioning}%
2272   {\BabelPatchSection{part}%
2273    \BabelPatchSection{chapter}%
2274    \BabelPatchSection{section}%
2275    \BabelPatchSection{subsection}%
2276    \BabelPatchSection{subsubsection}%
2277    \BabelPatchSection{paragraph}%
2278    \BabelPatchSection{subparagraph}%
2279    \def\babel@toc#1{%
2280      \select@language@x{\bbl@main@language}}}{}
2281 \IfBabelLayout{captions}%
2282   {\BabelPatchSection{caption}}{}
```

7.14 Load engine specific macros

Some macros are not defined in all engines, so, after loading the files define them if necessary to raise an error.

```
2283 \bbl@trace{Input engine specific macros}
2284 \ifcase\bbl@engine
2285   \input txtbabel.def
2286 \or
2287   \input luababel.def
2288 \or
2289   \input xebabel.def
2290 \fi
2291 \providecommand\babelfont{%
2292   \bbl@error
2293     {This macro is available only in LuaLaTeX and XeLaTeX.}%
2294     {Consider switching to these engines.}}
2295 \providecommand\babelprehyphenation{%
2296   \bbl@error
2297     {This macro is available only in LuaLaTeX.}%
2298     {Consider switching to that engine.}}
2299 \ifx\babelposthyphenation\@undefined
2300   \let\babelposthyphenation\babelprehyphenation
2301   \let\babelpatterns\babelprehyphenation
2302   \let\babelcharproperty\babelprehyphenation
2303 \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 previouly loaded ldf files.

```
2304 \bbl@trace{Creating languages and reading ini files}
2305 \let\bbl@extend@ini\@gobble
2306 \newcommand\babelprovide[2][]{%
2307   \let\bbl@savelangname\languagename
2308   \edef\bbl@savelocaleid{\the\localeid}%
2309   % Set name and locale id
2310   \edef\languagename{#2}%
```

```
2311    \bbl@id@assign
2312    % Initialize keys
2313    \bbl@vforeach{captions,date,import,main,script,language,%
2314        hyphenrules,linebreaking,justification,mapfont,maparabic,%
2315        mapdigits,intraspace,intrapenalty,onchar,transforms,alph,%
2316        Alph,labels,labels*,calendar,date}%
2317      {\bbl@csarg\let{KVP@##1}\@nnil}%
2318    \global\let\bbl@release@transforms\@empty
2319    \let\bbl@calendars\@empty
2320    \global\let\bbl@inidata\@empty
2321    \global\let\bbl@extend@ini\@gobble
2322    \gdef\bbl@key@list{;}%
2323    \bbl@forkv{#1}{%
2324      \in@{/}{##1}%
2325      \ifin@
2326        \global\let\bbl@extend@ini\bbl@extend@ini@aux
2327        \bbl@renewinikey##1\@@{##2}%
2328      \else
2329        \bbl@csarg\ifx{KVP@##1}\@nnil\else
2330          \bbl@error
2331            {Unknown key '##1' in \string\babelprovide}%
2332            {See the manual for valid keys}%
2333        \fi
2334        \bbl@csarg\def{KVP@##1}{##2}%
2335      \fi}%
2336    \chardef\bbl@howloaded=% 0:none; 1:ldf without ini; 2:ini
2337      \bbl@ifunset{date#2}\z@{\bbl@ifunset{bbl@llevel@#2}\@ne\tw@}%
2338    % == init ==
2339    \ifx\bbl@screset\@undefined
2340      \bbl@ldfinit
2341    \fi
2342    % == date (as option) ==
2343    % \ifx\bbl@KVP@date\@nnil\else
2344    % \fi
2345    % ==
2346    \let\bbl@lbkflag\relax % \@empty = do setup linebreak
2347    \ifcase\bbl@howloaded
2348      \let\bbl@lbkflag\@empty % new
2349    \else
2350      \ifx\bbl@KVP@hyphenrules\@nnil\else
2351         \let\bbl@lbkflag\@empty
2352      \fi
2353      \ifx\bbl@KVP@import\@nnil\else
2354        \let\bbl@lbkflag\@empty
2355      \fi
2356    \fi
2357    % == import, captions ==
2358    \ifx\bbl@KVP@import\@nnil\else
2359      \bbl@exp{\\\bbl@ifblank{\bbl@KVP@import}}%
2360        {\ifx\bbl@initoload\relax
2361           \begingroup
2362             \def\BabelBeforeIni##1##2{\gdef\bbl@KVP@import{##1}\endinput}%
2363             \bbl@input@texini{#2}%
2364           \endgroup
2365         \else
2366           \xdef\bbl@KVP@import{\bbl@initoload}%
2367         \fi}%
2368        {}%
2369      \let\bbl@KVP@date\@empty
2370    \fi
2371    \ifx\bbl@KVP@captions\@nnil
2372      \let\bbl@KVP@captions\bbl@KVP@import
2373    \fi
```

```
2374  % ==
2375  \ifx\bbl@KVP@transforms\@nnil\else
2376    \bbl@replace\bbl@KVP@transforms{ }{,}%
2377  \fi
2378  % == Load ini ==
2379  \ifcase\bbl@howloaded
2380    \bbl@provide@new{#2}%
2381  \else
2382    \bbl@ifblank{#1}%
2383      {}%  With \bbl@load@basic below
2384      {\bbl@provide@renew{#2}}%
2385  \fi
2386  % Post tasks
2387  % ----------
2388  % == subsequent calls after the first provide for a locale ==
2389  \ifx\bbl@inidata\@empty\else
2390    \bbl@extend@ini{#2}%
2391  \fi
2392  % == ensure captions ==
2393  \ifx\bbl@KVP@captions\@nnil\else
2394    \bbl@ifunset{bbl@extracaps@#2}%
2395      {\bbl@exp{\\\babelensure[exclude=\\\today]{#2}}}%
2396      {\bbl@exp{\\\babelensure[exclude=\\\today,
2397                include=\[bbl@extracaps@#2]]{#2}}}%
2398    \bbl@ifunset{bbl@ensure@\languagename}%
2399      {\bbl@exp{%
2400        \\\DeclareRobustCommand\<bbl@ensure@\languagename>[1]{%
2401          \\\foreignlanguage{\languagename}%
2402          {####1}}}}%
2403      {}%
2404    \bbl@exp{%
2405      \\\bbl@toglobal\<bbl@ensure@\languagename>%
2406      \\\bbl@toglobal\<bbl@ensure@\languagename\space>}%
2407  \fi
2408  % ==
2409  % At this point all parameters are defined if 'import'. Now we
2410  % execute some code depending on them. But what about if nothing was
2411  % imported? We just set the basic parameters, but still loading the
2412  % whole ini file.
2413  \bbl@load@basic{#2}%
2414  % == script, language ==
2415  % Override the values from ini or defines them
2416  \ifx\bbl@KVP@script\@nnil\else
2417    \bbl@csarg\edef{sname@#2}{\bbl@KVP@script}%
2418  \fi
2419  \ifx\bbl@KVP@language\@nnil\else
2420    \bbl@csarg\edef{lname@#2}{\bbl@KVP@language}%
2421  \fi
2422  \ifcase\bbl@engine\or
2423    \bbl@ifunset{bbl@chrng@\languagename}{}%
2424      {\directlua{
2425        Babel.set_chranges_b('\bbl@cl{sbcp}', '\bbl@cl{chrng}') }}%
2426  \fi
2427   % == onchar ==
2428  \ifx\bbl@KVP@onchar\@nnil\else
2429    \bbl@luahyphenate
2430    \bbl@exp{%
2431      \\\AddToHook{env/document/before}{{\\\select@language{#2}{}}}}%
2432    \directlua{
2433      if Babel.locale_mapped == nil then
2434        Babel.locale_mapped = true
2435        Babel.linebreaking.add_before(Babel.locale_map)
2436        Babel.loc_to_scr = {}
```

```
2437        Babel.chr_to_loc = Babel.chr_to_loc or {}
2438      end}%
2439    \bbl@xin@{ ids }{ \bbl@KVP@onchar\space}%
2440    \ifin@
2441      \ifx\bbl@starthyphens\@undefined % Needed if no explicit selection
2442        \AddBabelHook{babel-onchar}{beforestart}{{\bbl@starthyphens}}%
2443      \fi
2444      \bbl@exp{\\\bbl@add\\\bbl@starthyphens
2445        {\\\bbl@patterns@lua{\languagename}}}%
2446      % TODO - error/warning if no script
2447      \directlua{
2448        if Babel.script_blocks['\bbl@cl{sbcp}'] then
2449          Babel.loc_to_scr[\the\localeid] =
2450            Babel.script_blocks['\bbl@cl{sbcp}']
2451          Babel.locale_props[\the\localeid].lc = \the\localeid\space
2452          Babel.locale_props[\the\localeid].lg = \the\@nameuse{l@\languagename}\space
2453        end
2454      }%
2455    \fi
2456    \bbl@xin@{ fonts }{ \bbl@KVP@onchar\space}%
2457    \ifin@
2458      \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
2459      \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
2460      \directlua{
2461        if Babel.script_blocks['\bbl@cl{sbcp}'] then
2462          Babel.loc_to_scr[\the\localeid] =
2463            Babel.script_blocks['\bbl@cl{sbcp}']
2464      end}%
2465      \ifx\bbl@mapselect\@undefined  % TODO. almost the same as mapfont
2466        \AtBeginDocument{%
2467          \bbl@patchfont{{\bbl@mapselect}}%
2468          {\selectfont}}%
2469        \def\bbl@mapselect{%
2470          \let\bbl@mapselect\relax
2471          \edef\bbl@prefontid{\fontid\font}}%
2472        \def\bbl@mapdir##1{%
2473          {\def\languagename{##1}%
2474          \let\bbl@ifrestoring\@firstoftwo % To avoid font warning
2475          \bbl@switchfont
2476          \ifnum\fontid\font>\z@ % A hack, for the pgf nullfont hack
2477            \directlua{
2478              Babel.locale_props[\the\csname bbl@id@@##1\endcsname]%
2479                     ['/\bbl@prefontid'] = \fontid\font\space}%
2480          \fi}}%
2481      \fi
2482      \bbl@exp{\\\bbl@add\\\bbl@mapselect{\\\bbl@mapdir{\languagename}}}%
2483    \fi
2484    % TODO - catch non-valid values
2485    \fi
2486  % == mapfont ==
2487  % For bidi texts, to switch the font based on direction
2488  \ifx\bbl@KVP@mapfont\@nnil\else
2489    \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}{}%
2490      {\bbl@error{Option '\bbl@KVP@mapfont' unknown for\\%
2491                  mapfont. Use 'direction'.%
2492                {See the manual for details.}}}%
2493    \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
2494    \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
2495    \ifx\bbl@mapselect\@undefined % TODO. See onchar.
2496      \AtBeginDocument{%
2497        \bbl@patchfont{{\bbl@mapselect}}%
2498        {\selectfont}}%
2499      \def\bbl@mapselect{%
```

```
2500        \let\bbl@mapselect\relax
2501        \edef\bbl@prefontid{\fontid\font}}%
2502      \def\bbl@mapdir##1{%
2503        {\def\languagename{##1}%
2504         \let\bbl@ifrestoring\@firstoftwo % avoid font warning
2505         \bbl@switchfont
2506         \directlua{Babel.fontmap
2507           [\the\csname bbl@wdir@##1\endcsname]%
2508           [\bbl@prefontid]=\fontid\font}}}%
2509    \fi
2510    \bbl@exp{\\\bbl@add\\\bbl@mapselect{\\\bbl@mapdir{\languagename}}}%
2511  \fi
2512  % == Line breaking: intraspace, intrapenalty ==
2513  % For CJK, East Asian, Southeast Asian, if interspace in ini
2514  \ifx\bbl@KVP@intraspace\@nnil\else % We can override the ini or set
2515    \bbl@csarg\edef{intsp@#2}{\bbl@KVP@intraspace}%
2516  \fi
2517  \bbl@provide@intraspace
2518  % == Line breaking: CJK quotes ==
2519  \ifcase\bbl@engine\or
2520    \bbl@xin@{/c}{/\bbl@cl{lnbrk}}%
2521    \ifin@
2522      \bbl@ifunset{bbl@quote@\languagename}{}%
2523        {\directlua{
2524           Babel.locale_props[\the\localeid].cjk_quotes = {}
2525           local cs = 'op'
2526           for c in string.utfvalues(%
2527               [[\csname bbl@quote@\languagename\endcsname]]) do
2528             if Babel.cjk_characters[c].c == 'qu' then
2529               Babel.locale_props[\the\localeid].cjk_quotes[c] = cs
2530             end
2531             cs = ( cs == 'op') and 'cl' or 'op'
2532           end
2533        }}%
2534    \fi
2535  \fi
2536  % == Line breaking: justification ==
2537  \ifx\bbl@KVP@justification\@nnil\else
2538    \let\bbl@KVP@linebreaking\bbl@KVP@justification
2539  \fi
2540  \ifx\bbl@KVP@linebreaking\@nnil\else
2541    \bbl@xin@{,\bbl@KVP@linebreaking,}{,elongated,kashida,cjk,unhyphenated,}%
2542    \ifin@
2543      \bbl@csarg\xdef
2544        {lnbrk@\languagename}{\expandafter\@car\bbl@KVP@linebreaking\@nil}%
2545    \fi
2546  \fi
2547  \bbl@xin@{/e}{/\bbl@cl{lnbrk}}%
2548  \ifin@\else\bbl@xin@{/k}{/\bbl@cl{lnbrk}}\fi
2549  \ifin@\bbl@arabicjust\fi
2550  % == Line breaking: hyphenate.other.(locale|script) ==
2551  \ifx\bbl@lbkflag\@empty
2552    \bbl@ifunset{bbl@hyotl@\languagename}{}%
2553      {\bbl@csarg\bbl@replace{hyotl@\languagename}{ }{,}%
2554       \bbl@startcommands*{\languagename}{}%
2555         \bbl@csarg\bbl@foreach{hyotl@\languagename}{%
2556           \ifcase\bbl@engine
2557             \ifnum##1<257
2558               \SetHyphenMap{\BabelLower{##1}{##1}}%
2559             \fi
2560           \else
2561             \SetHyphenMap{\BabelLower{##1}{##1}}%
2562           \fi}%
```

116

```
2563            \bbl@endcommands}%
2564      \bbl@ifunset{bbl@hyots@\languagename}{}%
2565        {\bbl@csarg\bbl@replace{hyots@\languagename}{ }{,}%
2566          \bbl@csarg\bbl@foreach{hyots@\languagename}{%
2567            \ifcase\bbl@engine
2568              \ifnum##1<257
2569                \global\lccode##1=##1\relax
2570              \fi
2571            \else
2572              \global\lccode##1=##1\relax
2573            \fi}}%
2574  \fi
2575  % == Counters: maparabic ==
2576  % Native digits, if provided in ini (TeX level, xe and lua)
2577  \ifcase\bbl@engine\else
2578    \bbl@ifunset{bbl@dgnat@\languagename}{}%
2579      {\expandafter\ifx\csname bbl@dgnat@\languagename\endcsname\@empty\else
2580        \expandafter\expandafter\expandafter
2581        \bbl@setdigits\csname bbl@dgnat@\languagename\endcsname
2582        \ifx\bbl@KVP@maparabic\@nnil\else
2583          \ifx\bbl@latinarabic\@undefined
2584            \expandafter\let\expandafter\@arabic
2585              \csname bbl@counter@\languagename\endcsname
2586          \else    % ie, if layout=counters, which redefines \@arabic
2587            \expandafter\let\expandafter\bbl@latinarabic
2588              \csname bbl@counter@\languagename\endcsname
2589          \fi
2590        \fi
2591      \fi}%
2592  \fi
2593  % == Counters: mapdigits ==
2594  % Native digits (lua level).
2595  \ifodd\bbl@engine
2596    \ifx\bbl@KVP@mapdigits\@nnil\else
2597      \bbl@ifunset{bbl@dgnat@\languagename}{}%
2598        {\RequirePackage{luatexbase}%
2599          \bbl@activate@preotf
2600          \directlua{
2601            Babel = Babel or {}  %%% -> presets in luababel
2602            Babel.digits_mapped = true
2603            Babel.digits = Babel.digits or {}
2604            Babel.digits[\the\localeid] =
2605              table.pack(string.utfvalue('\bbl@cl{dgnat}'))
2606            if not Babel.numbers then
2607              function Babel.numbers(head)
2608                local LOCALE = Babel.attr_locale
2609                local GLYPH = node.id'glyph'
2610                local inmath = false
2611                for item in node.traverse(head) do
2612                  if not inmath and item.id == GLYPH then
2613                    local temp = node.get_attribute(item, LOCALE)
2614                    if Babel.digits[temp] then
2615                      local chr = item.char
2616                      if chr > 47 and chr < 58 then
2617                        item.char = Babel.digits[temp][chr-47]
2618                      end
2619                    end
2620                  elseif item.id == node.id'math' then
2621                    inmath = (item.subtype == 0)
2622                  end
2623                end
2624                return head
2625              end
```

```
2626            end
2627          }}%
2628      \fi
2629    \fi
2630    % == Counters: alph, Alph ==
2631    % What if extras<lang> contains a \babel@save\@alph? It won't be
2632    % restored correctly when exiting the language, so we ignore
2633    % this change with the \bbl@alph@saved trick.
2634    \ifx\bbl@KVP@alph\@nnil\else
2635      \bbl@extras@wrap{\\\bbl@alph@saved}%
2636        {\let\bbl@alph@saved\@alph}%
2637        {\let\@alph\bbl@alph@saved
2638         \babel@save\@alph}%
2639      \bbl@exp{%
2640        \\\bbl@add\<extras\languagename>{%
2641          \let\\\@alph\<bbl@cntr@\bbl@KVP@alph @\languagename>}}%
2642    \fi
2643    \ifx\bbl@KVP@Alph\@nnil\else
2644      \bbl@extras@wrap{\\\bbl@Alph@saved}%
2645        {\let\bbl@Alph@saved\@Alph}%
2646        {\let\@Alph\bbl@Alph@saved
2647         \babel@save\@Alph}%
2648      \bbl@exp{%
2649        \\\bbl@add\<extras\languagename>{%
2650          \let\\\@Alph\<bbl@cntr@\bbl@KVP@Alph @\languagename>}}%
2651    \fi
2652    % == Calendars ==
2653    \ifx\bbl@KVP@calendar\@nnil
2654      \edef\bbl@KVP@calendar{\bbl@cl{calpr}}%
2655    \fi
2656    \def\bbl@tempe##1 ##2\@@{% Get first calendar
2657      \def\bbl@tempa{##1}}%
2658      \bbl@exp{\\\bbl@tempe\bbl@KVP@calendar\space\\\@@}%
2659    \def\bbl@tempe##1.##2.##3\@@{%
2660      \def\bbl@tempc{##1}%
2661      \def\bbl@tempb{##2}}%
2662    \expandafter\bbl@tempe\bbl@tempa..\@@
2663    \bbl@csarg\edef{calpr@\languagename}{%
2664      \ifx\bbl@tempc\@empty\else
2665        calendar=\bbl@tempc
2666      \fi
2667      \ifx\bbl@tempb\@empty\else
2668        ,variant=\bbl@tempb
2669      \fi}%
2670    % == require.babel in ini ==
2671    % To load or reaload the babel-*.tex, if require.babel in ini
2672    \ifx\bbl@beforestart\relax\else  % But not in doc aux or body
2673      \bbl@ifunset{bbl@rqtex@\languagename}{}%
2674        {\expandafter\ifx\csname bbl@rqtex@\languagename\endcsname\@empty\else
2675          \let\BabelBeforeIni\@gobbletwo
2676          \chardef\atcatcode=\catcode`\@
2677          \catcode`\@=11\relax
2678          \bbl@input@texini{\bbl@cs{rqtex@\languagename}}%
2679          \catcode`\@=\atcatcode
2680          \let\atcatcode\relax
2681          \global\bbl@csarg\let{rqtex@\languagename}\relax
2682        \fi}%
2683      \bbl@foreach\bbl@calendars{%
2684        \bbl@ifunset{bbl@ca@##1}{%
2685          \chardef\atcatcode=\catcode`\@
2686          \catcode`\@=11\relax
2687          \InputIfFileExists{babel-ca-##1.tex}{}{}%
2688          \catcode`\@=\atcatcode
```

118

```
2689            \let\atcatcode\relax}%
2690        {}}%
2691   \fi
2692   % == frenchspacing ==
2693   \ifcase\bbl@howloaded\in@true\else\in@false\fi
2694   \ifin@\else\bbl@xin@{typography/frenchspacing}{\bbl@key@list}\fi
2695   \ifin@
2696      \bbl@extras@wrap{\\\bbl@pre@fs}%
2697        {\bbl@pre@fs}%
2698        {\bbl@post@fs}%
2699   \fi
2700   % == Release saved transforms ==
2701   \bbl@release@transforms\relax % \relax closes the last item.
2702   % == main ==
2703   \ifx\bbl@KVP@main\@nnil  % Restore only if not 'main'
2704      \let\languagename\bbl@savelangname
2705      \chardef\localeid\bbl@savelocaleid\relax
2706   \fi}
```

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

```
2707 \def\bbl@provide@new#1{%
2708   \@namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
2709   \@namedef{extras#1}{}%
2710   \@namedef{noextras#1}{}%
2711   \bbl@startcommands*{#1}{captions}%
2712      \ifx\bbl@KVP@captions\@nnil %        and also if import, implicit
2713        \def\bbl@tempb##1{%              elt for \bbl@captionslist
2714          \ifx##1\@empty\else
2715            \bbl@exp{%
2716              \\\SetString\\##1{%
2717                \\\bbl@nocaption{\bbl@stripslash##1}{#1\bbl@stripslash##1}}}%
2718            \expandafter\bbl@tempb
2719          \fi}%
2720        \expandafter\bbl@tempb\bbl@captionslist\@empty
2721      \else
2722        \ifx\bbl@initoload\relax
2723          \bbl@read@ini{\bbl@KVP@captions}2%  % Here letters cat = 11
2724        \else
2725          \bbl@read@ini{\bbl@initoload}2%      % Same
2726        \fi
2727      \fi
2728   \StartBabelCommands*{#1}{date}%
2729      \ifx\bbl@KVP@date\@nnil
2730        \bbl@exp{%
2731          \\\SetString\\\today{\\\bbl@nocaption{today}{#1today}}}%
2732      \else
2733        \bbl@savetoday
2734        \bbl@savedate
2735      \fi
2736   \bbl@endcommands
2737   \bbl@load@basic{#1}%
2738   % == hyphenmins == (only if new)
2739   \bbl@exp{%
2740      \gdef\<#1hyphenmins>{%
2741        {\bbl@ifunset{bbl@lfthm@#1}{2}{\bbl@cs{lfthm@#1}}}%
2742        {\bbl@ifunset{bbl@rgthm@#1}{3}{\bbl@cs{rgthm@#1}}}}}%
2743   % == hyphenrules (also in renew) ==
2744   \bbl@provide@hyphens{#1}%
2745   \ifx\bbl@KVP@main\@nnil\else
2746      \expandafter\main@language\expandafter{#1}%
2747   \fi}
2748 %
```

```
2749 \def\bbl@provide@renew#1{%
2750   \ifx\bbl@KVP@captions\@nnil\else
2751     \StartBabelCommands*{#1}{captions}%
2752       \bbl@read@ini{\bbl@KVP@captions}2%   % Here all letters cat = 11
2753     \EndBabelCommands
2754   \fi
2755   \ifx\bbl@KVP@date\@nnil\else
2756     \StartBabelCommands*{#1}{date}%
2757       \bbl@savetoday
2758       \bbl@savedate
2759     \EndBabelCommands
2760   \fi
2761   % == hyphenrules (also in new) ==
2762   \ifx\bbl@lbkflag\@empty
2763     \bbl@provide@hyphens{#1}%
2764   \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.)

```
2765 \def\bbl@load@basic#1{%
2766   \ifcase\bbl@howloaded\or\or
2767     \ifcase\csname bbl@llevel@\languagename\endcsname
2768       \bbl@csarg\let{lname@\languagename}\relax
2769     \fi
2770   \fi
2771   \bbl@ifunset{bbl@lname@#1}%
2772     {\def\BabelBeforeIni##1##2{%
2773       \begingroup
2774         \let\bbl@ini@captions@aux\@gobbletwo
2775         \def\bbl@inidate ####1.####2.####3.####4\relax ####5####6{}%
2776         \bbl@read@ini{##1}1%
2777         \ifx\bbl@initoload\relax\endinput\fi
2778       \endgroup}%
2779     \begingroup        % boxed, to avoid extra spaces:
2780       \ifx\bbl@initoload\relax
2781         \bbl@input@texini{#1}%
2782       \else
2783         \setbox\z@\hbox{\BabelBeforeIni{\bbl@initoload}{}}%
2784       \fi
2785     \endgroup}%
2786     {}}
```

The hyphenrules option is handled with an auxiliary macro.

```
2787 \def\bbl@provide@hyphens#1{%
2788   \let\bbl@tempa\relax
2789   \ifx\bbl@KVP@hyphenrules\@nnil\else
2790     \bbl@replace\bbl@KVP@hyphenrules{ }{,}%
2791     \bbl@foreach\bbl@KVP@hyphenrules{%
2792       \ifx\bbl@tempa\relax    % if not yet found
2793         \bbl@ifsamestring{##1}{+}%
2794           {{\bbl@exp{\\\addlanguage\<l@##1>}}}%
2795           {}%
2796         \bbl@ifunset{l@##1}%
2797           {}%
2798           {\bbl@exp{\let\bbl@tempa\<l@##1>}}%
2799       \fi}%
2800   \fi
2801   \ifx\bbl@tempa\relax %         if no opt or no language in opt found
2802     \ifx\bbl@KVP@import\@nnil
2803       \ifx\bbl@initoload\relax\else
2804         \bbl@exp{%             and hyphenrules is not empty
2805           \\\bbl@ifblank{\bbl@cs{hyphr@#1}}%
2806             {}%
```

```
2807                {\let\\\bbl@tempa\<l@\bbl@cl{hyphr}>}}%
2808          \fi
2809      \else % if importing
2810        \bbl@exp{%                          and hyphenrules is not empty
2811          \\\bbl@ifblank{\bbl@cs{hyphr@#1}}%
2812            {}%
2813            {\let\\\bbl@tempa\<l@\bbl@cl{hyphr}>}}%
2814      \fi
2815    \fi
2816    \bbl@ifunset{bbl@tempa}%          ie, relax or undefined
2817      {\bbl@ifunset{l@#1}%              no hyphenrules found - fallback
2818        {\bbl@exp{\\\adddialect\<l@#1>\language}}%
2819        {}}%                            so, l@<lang> is ok - nothing to do
2820      {\bbl@exp{\\\adddialect\<l@#1>\bbl@tempa}}}% found in opt list or ini
```

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

```
2821 \def\bbl@input@texini#1{%
2822    \bbl@bsphack
2823      \bbl@exp{%
2824        \catcode`\\\%=14 \catcode`\\\\=0
2825        \catcode`\\\{=1  \catcode`\\\}=2
2826        \lowercase{\\\InputIfFileExists{babel-#1.tex}{}{}}%
2827        \catcode`\\\%=\the\catcode`\%\relax
2828        \catcode`\\\\=\the\catcode`\\\relax
2829        \catcode`\\\{=\the\catcode`\{\relax
2830        \catcode`\\\}=\the\catcode`\}\relax}%
2831    \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`.

```
2832 \def\bbl@iniline#1\bbl@iniline{%
2833    \@ifnextchar[\bbl@inisect{\@ifnextchar;\bbl@iniskip\bbl@inistore}#1\@@}% ]
2834 \def\bbl@inisect[#1]#2\@@{\def\bbl@section{#1}}
2835 \def\bbl@iniskip#1\@@{}%       if starts with ;
2836 \def\bbl@inistore#1=#2\@@{%      full (default)
2837    \bbl@trim@def\bbl@tempa{#1}%
2838    \bbl@trim\toks@{#2}%
2839    \bbl@xin@{;\bbl@section/\bbl@tempa;}{\bbl@key@list}%
2840    \ifin@\else
2841      \bbl@xin@{,identification/include.}%
2842              {,\bbl@section/\bbl@tempa}%
2843      \ifin@\edef\bbl@required@inis{\the\toks@}\fi
2844      \bbl@exp{%
2845        \\\g@addto@macro\\\bbl@inidata{%
2846          \\\bbl@elt{\bbl@section}{\bbl@tempa}{\the\toks@}}}%
2847    \fi}
2848 \def\bbl@inistore@min#1=#2\@@{%  minimal (maybe set in \bbl@read@ini)
2849    \bbl@trim@def\bbl@tempa{#1}%
2850    \bbl@trim\toks@{#2}%
2851    \bbl@xin@{.identification.}{.\bbl@section.}%
2852    \ifin@
2853      \bbl@exp{\\\g@addto@macro\\\bbl@inidata{%
2854        \\\bbl@elt{identification}{\bbl@tempa}{\the\toks@}}}%
2855    \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.

```
2856 \def\bbl@loop@ini{%
2857    \loop
```

```
2858    \if T\ifeof\bbl@readstream F\fi T\relax % Trick, because inside \loop
2859      \endlinechar\m@ne
2860      \read\bbl@readstream to \bbl@line
2861      \endlinechar`\^^M
2862      \ifx\bbl@line\@empty\else
2863        \expandafter\bbl@iniline\bbl@line\bbl@iniline
2864      \fi
2865    \repeat}
2866 \ifx\bbl@readstream\@undefined
2867   \csname newread\endcsname\bbl@readstream
2868 \fi
2869 \def\bbl@read@ini#1#2{%
2870   \global\let\bbl@extend@ini\@gobble
2871   \openin\bbl@readstream=babel-#1.ini
2872   \ifeof\bbl@readstream
2873     \bbl@error
2874       {There is no ini file for the requested language\\%
2875        (#1: \languagename). Perhaps you misspelled it or your\\%
2876        installation is not complete.}%
2877       {Fix the name or reinstall babel.}%
2878   \else
2879     % == Store ini data in \bbl@inidata ==
2880     \catcode`\[=12 \catcode`\]=12 \catcode`\==12 \catcode`\&=12
2881     \catcode`\;=12 \catcode`\|=12 \catcode`\%=14 \catcode`\-=12
2882     \bbl@info{Importing
2883                 \ifcase#2font and identification \or basic \fi
2884                  data for \languagename\\%
2885               from babel-#1.ini. Reported}%
2886     \ifnum#2=\z@
2887       \global\let\bbl@inidata\@empty
2888       \let\bbl@inistore\bbl@inistore@min   % Remember it's local
2889     \fi
2890     \def\bbl@section{identification}%
2891     \let\bbl@required@inis\@empty
2892     \bbl@exp{\\\bbl@inistore tag.ini=#1\\\@@}%
2893     \bbl@inistore load.level=#2\@@
2894     \bbl@loop@ini
2895     \ifx\bbl@required@inis\@empty\else
2896       \bbl@replace\bbl@required@inis{ }{,}%
2897       \bbl@foreach\bbl@required@inis{%
2898         \openin\bbl@readstream=##1.ini
2899         \bbl@loop@ini}%
2900     \fi
2901     % == Process stored data ==
2902     \bbl@csarg\xdef{lini@\languagename}{#1}%
2903     \bbl@read@ini@aux
2904     % == 'Export' data ==
2905     \bbl@ini@exports{#2}%
2906     \global\bbl@csarg\let{inidata@\languagename}\bbl@inidata
2907     \global\let\bbl@inidata\@empty
2908     \bbl@exp{\\\bbl@add@list\\\bbl@ini@loaded{\languagename}}%
2909     \bbl@toglobal\bbl@ini@loaded
2910   \fi}
2911 \def\bbl@read@ini@aux{%
2912   \let\bbl@savestrings\@empty
2913   \let\bbl@savetoday\@empty
2914   \let\bbl@savedate\@empty
2915   \def\bbl@elt##1##2##3{%
2916     \def\bbl@section{##1}%
2917     \in@{=date.}{=##1}% Find a better place
2918     \ifin@
2919       \bbl@ifunset{bbl@inikv@##1}%
2920         {\bbl@ini@calendar{##1}}%
```

```
2921        {}%
2922     \fi
2923     \in@{=identification/extension.}{=##1/##2}%
2924     \ifin@
2925       \bbl@ini@extension{##2}%
2926     \fi
2927     \bbl@ifunset{bbl@inikv@##1}{}%
2928       {\csname bbl@inikv@##1\endcsname{##2}{##3}}}%
2929   \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.

```
2930 \def\bbl@extend@ini@aux#1{%
2931   \bbl@startcommands*{#1}{captions}%
2932     % Activate captions/... and modify exports
2933     \bbl@csarg\def{inikv@captions.licr}##1##2{%
2934       \setlocalecaption{#1}{##1}{##2}}%
2935     \def\bbl@inikv@captions##1##2{%
2936       \bbl@ini@captions@aux{##1}{##2}}%
2937     \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2938     \def\bbl@exportkey##1##2##3{%
2939       \bbl@ifunset{bbl@@kv@##2}{}%
2940         {\expandafter\ifx\csname bbl@@kv@##2\endcsname\@empty\else
2941           \bbl@exp{\global\let\<bbl@##1@\languagename>\<bbl@@kv@##2>}%
2942         \fi}}%
2943     % As with \bbl@read@ini, but with some changes
2944     \bbl@read@ini@aux
2945     \bbl@ini@exports\tw@
2946     % Update inidata@lang by pretending the ini is read.
2947     \def\bbl@elt##1##2##3{%
2948       \def\bbl@section{##1}%
2949       \bbl@iniline##2=##3\bbl@iniline}%
2950     \csname bbl@inidata@#1\endcsname
2951     \global\bbl@csarg\let{inidata@#1}\bbl@inidata
2952   \StartBabelCommands*{#1}{date}% And from the import stuff
2953     \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2954     \bbl@savetoday
2955     \bbl@savedate
2956   \bbl@endcommands}
```

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

```
2957 \def\bbl@ini@calendar#1{%
2958   \lowercase{\def\bbl@tempa{=#1=}}%
2959   \bbl@replace\bbl@tempa{=date.gregorian}{}%
2960   \bbl@replace\bbl@tempa{=date.}{}%
2961   \in@{.licr=}{#1=}%
2962   \ifin@
2963     \ifcase\bbl@engine
2964       \bbl@replace\bbl@tempa{.licr=}{}%
2965     \else
2966       \let\bbl@tempa\relax
2967     \fi
2968   \fi
2969   \ifx\bbl@tempa\relax\else
2970     \bbl@replace\bbl@tempa{=}{}%
2971     \ifx\bbl@tempa\@empty\else
2972       \xdef\bbl@calendars{\bbl@calendars,\bbl@tempa}%
2973     \fi
2974     \bbl@exp{%
2975       \def\<bbl@inikv@#1>####1####2{%
2976         \\\bbl@inidate####1...\relax{####2}{\bbl@tempa}}}%
2977 \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).

```
2978 \def\bbl@renewinikey#1/#2\@@#3{%
2979   \edef\bbl@tempa{\zap@space #1 \@empty}%    section
2980   \edef\bbl@tempb{\zap@space #2 \@empty}%    key
2981   \bbl@trim\toks@{#3}%                       value
2982   \bbl@exp{%
2983     \edef\\\bbl@key@list{\bbl@key@list \bbl@tempa/\bbl@tempb;}%
2984     \\\g@addto@macro\\\bbl@inidata{%
2985       \\\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.

```
2986 \def\bbl@exportkey#1#2#3{%
2987   \bbl@ifunset{bbl@@kv@#2}%
2988     {\bbl@csarg\gdef{#1@\languagename}{#3}}%
2989     {\expandafter\ifx\csname bbl@@kv@#2\endcsname\@empty
2990       \bbl@csarg\gdef{#1@\languagename}{#3}%
2991      \else
2992       \bbl@exp{\global\let\<bbl@#1@\languagename>\<bbl@@kv@#2>}%
2993     \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.

```
2994 \def\bbl@iniwarning#1{%
2995   \bbl@ifunset{bbl@@kv@identification.warning#1}{}%
2996     {\bbl@warning{%
2997       From babel-\bbl@cs{lini@\languagename}.ini:\\%
2998       \bbl@cs{@kv@identification.warning#1}\\%
2999       Reported }}}
3000 %
3001 \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.

```
3002 \def\bbl@ini@extension#1{%
3003   \def\bbl@tempa{#1}%
3004   \bbl@replace\bbl@tempa{extension.}{}%
3005   \bbl@replace\bbl@tempa{.tag.bcp47}{}%
3006   \bbl@ifunset{bbl@info@#1}%
3007     {\bbl@csarg\xdef{info@#1}{ext/\bbl@tempa}%
3008      \bbl@exp{%
3009        \\\g@addto@macro\\\bbl@moreinfo{%
3010          \\\bbl@exportkey{ext/\bbl@tempa}{identification.#1}{}}}}%
3011     {}}
3012 \let\bbl@moreinfo\@empty
3013 %
3014 \def\bbl@ini@exports#1{%
3015   % Identification always exported
3016   \bbl@iniwarning{}%
3017   \ifcase\bbl@engine
3018     \bbl@iniwarning{.pdflatex}%
3019   \or
3020     \bbl@iniwarning{.lualatex}%
3021   \or
3022     \bbl@iniwarning{.xelatex}%
3023   \fi%
3024   \bbl@exportkey{llevel}{identification.load.level}{}%
3025   \bbl@exportkey{elname}{identification.name.english}{}%
3026   \bbl@exp{\\\bbl@exportkey{lname}{identification.name.opentype}%
3027     {\csname bbl@elname@\languagename\endcsname}}%
3028   \bbl@exportkey{tbcp}{identification.tag.bcp47}{}%
```

```
3029  \bbl@exportkey{lbcp}{identification.language.tag.bcp47}{}%
3030  \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
3031  \bbl@exportkey{esname}{identification.script.name}{}%
3032  \bbl@exp{\\\bbl@exportkey{sname}{identification.script.name.opentype}%
3033    {\csname bbl@esname@\languagename\endcsname}}%
3034  \bbl@exportkey{sbcp}{identification.script.tag.bcp47}{}%
3035  \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
3036  \bbl@exportkey{rbcp}{identification.region.tag.bcp47}{}%
3037  \bbl@exportkey{vbcp}{identification.variant.tag.bcp47}{}%
3038  \bbl@moreinfo
3039  % Also maps bcp47 -> languagename
3040  \ifbbl@bcptoname
3041    \bbl@csarg\xdef{bcp@map@\bbl@cl{tbcp}}{\languagename}%
3042  \fi
3043  % Conditional
3044  \ifnum#1>\z@        % 0 = only info, 1, 2 = basic, (re)new
3045    \bbl@exportkey{calpr}{date.calendar.preferred}{}%
3046    \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
3047    \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
3048    \bbl@exportkey{lfthm}{typography.lefthyphenmin}{2}%
3049    \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
3050    \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
3051    \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
3052    \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
3053    \bbl@exportkey{intsp}{typography.intraspace}{}%
3054    \bbl@exportkey{frspc}{typography.frenchspacing}{u}%
3055    \bbl@exportkey{chrng}{characters.ranges}{}%
3056    \bbl@exportkey{quote}{characters.delimiters.quotes}{}%
3057    \bbl@exportkey{dgnat}{numbers.digits.native}{}%
3058    \ifnum#1=\tw@            % only (re)new
3059      \bbl@exportkey{rqtex}{identification.require.babel}{}%
3060      \bbl@toglobal\bbl@savetoday
3061      \bbl@toglobal\bbl@savedate
3062      \bbl@savestrings
3063    \fi
3064  \fi}
```

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

```
3065 \def\bbl@inikv#1#2{%        key=value
3066   \toks@{#2}%              This hides #'s from ini values
3067   \bbl@csarg\edef{@kv@\bbl@section.#1}{\the\toks@}}
```

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

```
3068 \let\bbl@inikv@identification\bbl@inikv
3069 \let\bbl@inikv@date\bbl@inikv
3070 \let\bbl@inikv@typography\bbl@inikv
3071 \let\bbl@inikv@characters\bbl@inikv
3072 \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'.

```
3073 \def\bbl@inikv@counters#1#2{%
3074   \bbl@ifsamestring{#1}{digits}%
3075     {\bbl@error{The counter name 'digits' is reserved for mapping\\%
3076                 decimal digits}%
3077                {Use another name.}}%
3078     {}%
3079   \def\bbl@tempc{#1}%
3080   \bbl@trim@def{\bbl@tempb*}{#2}%
3081   \in@{.1$}{#1$}%
3082   \ifin@
3083     \bbl@replace\bbl@tempc{.1}{}%
3084     \bbl@csarg\protected@xdef{cntr@\bbl@tempc @\languagename}{%
```

```
3085        \noexpand\bbl@alphnumeral{\bbl@tempc}}%
3086    \fi
3087    \in@{.F.}{#1}%
3088    \ifin@\else\in@{.S.}{#1}\fi
3089    \ifin@
3090      \bbl@csarg\protected@xdef{cntr@#1@\languagename}{\bbl@tempb*}%
3091    \else
3092      \toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa
3093      \expandafter\bbl@buildifcase\bbl@tempb* \\ % Space after \\
3094      \bbl@csarg{\global\expandafter\let}{cntr@#1@\languagename}\bbl@tempa
3095    \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.

```
3096 \ifcase\bbl@engine
3097  \bbl@csarg\def{inikv@captions.licr}#1#2{%
3098      \bbl@ini@captions@aux{#1}{#2}}
3099 \else
3100  \def\bbl@inikv@captions#1#2{%
3101      \bbl@ini@captions@aux{#1}{#2}}
3102 \fi
```

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

```
3103 \def\bbl@ini@captions@template#1#2{% string language tempa=capt-name
3104  \bbl@replace\bbl@tempa{.template}{}%
3105  \def\bbl@toreplace{#1{}}%
3106  \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
3107  \bbl@replace\bbl@toreplace{[[}{\csname}%
3108  \bbl@replace\bbl@toreplace{[}{\csname the}%
3109  \bbl@replace\bbl@toreplace{]]}{name\endcsname{}}%
3110  \bbl@replace\bbl@toreplace{]}{\endcsname{}}%
3111  \bbl@xin@{,\bbl@tempa,}{,chapter,appendix,part,}%
3112  \ifin@
3113    \@nameuse{bbl@patch\bbl@tempa}%
3114    \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
3115  \fi
3116  \bbl@xin@{,\bbl@tempa,}{,figure,table,}%
3117  \ifin@
3118    \toks@\expandafter{\bbl@toreplace}%
3119    \bbl@exp{\gdef\<fnum@\bbl@tempa>{\the\toks@}}%
3120  \fi}
3121 \def\bbl@ini@captions@aux#1#2{%
3122  \bbl@trim@def\bbl@tempa{#1}%
3123  \bbl@xin@{.template}{\bbl@tempa}%
3124  \ifin@
3125    \bbl@ini@captions@template{#2}\languagename
3126  \else
3127    \bbl@ifblank{#2}%
3128      {\bbl@exp{%
3129         \toks@{\\\bbl@nocaption{\bbl@tempa}{\languagename\bbl@tempa name}}}}%
3130      {\bbl@trim\toks@{#2}}%
3131    \bbl@exp{%
3132      \\\bbl@add\\\bbl@savestrings{%
3133        \\\SetString\<\bbl@tempa name>{\the\toks@}}}%
3134    \toks@\expandafter{\bbl@captionslist}%
3135    \bbl@exp{\\\in@{\<\bbl@tempa name>}{\the\toks@}}%
3136    \ifin@\else
3137      \bbl@exp{%
3138        \\\bbl@add\<bbl@extracaps@\languagename>{\<\bbl@tempa name>}%
3139        \\\bbl@toglobal\<bbl@extracaps@\languagename>}%
3140    \fi
3141  \fi}
```

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

```
3142 \def\bbl@list@the{%
3143   part,chapter,section,subsection,subsubsection,paragraph,%
3144   subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,%
3145   table,page,footnote,mpfootnote,mpfn}
3146 \def\bbl@map@cnt#1{%  #1:roman,etc, // #2:enumi,etc
3147   \bbl@ifunset{bbl@map@#1@\languagename}%
3148     {\@nameuse{#1}}%
3149     {\@nameuse{bbl@map@#1@\languagename}}}
3150 \def\bbl@inikv@labels#1#2{%
3151   \in@{.map}{#1}%
3152   \ifin@
3153     \ifx\bbl@KVP@labels\@nnil\else
3154       \bbl@xin@{ map }{ \bbl@KVP@labels\space}%
3155       \ifin@
3156         \def\bbl@tempc{#1}%
3157         \bbl@replace\bbl@tempc{.map}{}%
3158         \in@{,#2,}{,arabic,roman,Roman,alph,Alph,fnsymbol,}%
3159         \bbl@exp{%
3160           \gdef\<bbl@map@\bbl@tempc @\languagename>%
3161             {\ifin@\<#2>\else\\localecounter{#2}\fi}}%
3162         \bbl@foreach\bbl@list@the{%
3163           \bbl@ifunset{the##1}{}%
3164             {\bbl@exp{\let\\bbl@tempd\<the##1>}%
3165             \bbl@exp{%
3166               \\bbl@sreplace\<the##1>%
3167                 {\<\bbl@tempc>{##1}}{\\bbl@map@cnt{\bbl@tempc}{##1}}%
3168               \\bbl@sreplace\<the##1>%
3169                 {\<\@empty @\bbl@tempc>\<c@##1>}{\\bbl@map@cnt{\bbl@tempc}{##1}}}%
3170             \expandafter\ifx\csname the##1\endcsname\bbl@tempd\else
3171               \toks@\expandafter\expandafter\expandafter{%
3172                 \csname the##1\endcsname}%
3173               \expandafter\xdef\csname the##1\endcsname{{\the\toks@}}%
3174             \fi}}%
3175       \fi
3176     \fi
3177 %
3178 \else
3179   %
3180   % The following code is still under study. You can test it and make
3181   % suggestions. Eg, enumerate.2 = ([enumi]).([enumii]). It's
3182   % language dependent.
3183   \in@{enumerate.}{#1}%
3184   \ifin@
3185     \def\bbl@tempa{#1}%
3186     \bbl@replace\bbl@tempa{enumerate.}{}%
3187     \def\bbl@toreplace{#2}%
3188     \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
3189     \bbl@replace\bbl@toreplace{[}{\csname the}%
3190     \bbl@replace\bbl@toreplace{]}{\endcsname{}}%
3191     \toks@\expandafter{\bbl@toreplace}%
3192     % TODO. Execute only once:
3193     \bbl@exp{%
3194       \\bbl@add\<extras\languagename>{%
3195         \\babel@save\<labelenum\romannumeral\bbl@tempa>%
3196         \def\<labelenum\romannumeral\bbl@tempa>{\the\toks@}}%
3197       \\bbl@toglobal\<extras\languagename>}%
3198   \fi
3199 \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.

```
3200 \def\bbl@chaptype{chapter}
3201 \ifx\@makechapterhead\@undefined
3202   \let\bbl@patchchapter\relax
3203 \else\ifx\thechapter\@undefined
3204   \let\bbl@patchchapter\relax
3205 \else\ifx\ps@headings\@undefined
3206   \let\bbl@patchchapter\relax
3207 \else
3208   \def\bbl@patchchapter{%
3209     \global\let\bbl@patchchapter\relax
3210     \gdef\bbl@chfmt{%
3211       \bbl@ifunset{bbl@\bbl@chaptype fmt@\languagename}%
3212         {\@chapapp\space\thechapter}
3213         {\@nameuse{bbl@\bbl@chaptype fmt@\languagename}}}}
3214     \bbl@add\appendix{\def\bbl@chaptype{appendix}}% Not harmful, I hope
3215     \bbl@sreplace\ps@headings{\@chapapp\ \thechapter}{\bbl@chfmt}%
3216     \bbl@sreplace\chaptermark{\@chapapp\ \thechapter}{\bbl@chfmt}%
3217     \bbl@sreplace\@makechapterhead{\@chapapp\space\thechapter}{\bbl@chfmt}%
3218     \bbl@toglobal\appendix
3219     \bbl@toglobal\ps@headings
3220     \bbl@toglobal\chaptermark
3221     \bbl@toglobal\@makechapterhead}
3222   \let\bbl@patchappendix\bbl@patchchapter
3223 \fi\fi\fi
3224 \ifx\@part\@undefined
3225   \let\bbl@patchpart\relax
3226 \else
3227   \def\bbl@patchpart{%
3228     \global\let\bbl@patchpart\relax
3229     \gdef\bbl@partformat{%
3230       \bbl@ifunset{bbl@partfmt@\languagename}%
3231         {\partname\nobreakspace\thepart}
3232         {\@nameuse{bbl@partfmt@\languagename}}}}
3233     \bbl@sreplace\@part{\partname\nobreakspace\thepart}{\bbl@partformat}%
3234     \bbl@toglobal\@part}
3235 \fi
```

Date. Arguments (year, month, day) are *not* protected, on purpose. In \today, arguments are always gregorian, and therefore always converted with other calendars. TODO. Document

```
3236 \let\bbl@calendar\@empty
3237 \DeclareRobustCommand\localedate[1][]{\bbl@localedate{#1}}
3238 \def\bbl@localedate#1#2#3#4{%
3239   \begingroup
3240     \edef\bbl@they{#2}%
3241     \edef\bbl@them{#3}%
3242     \edef\bbl@thed{#4}%
3243     \edef\bbl@tempe{%
3244       \bbl@ifunset{bbl@calpr@\languagename}{}{\bbl@cl{calpr}},%
3245       #1}%
3246     \bbl@replace\bbl@tempe{ }{}%
3247     \bbl@replace\bbl@tempe{CONVERT}{convert=}% Hackish
3248     \bbl@replace\bbl@tempe{convert}{convert=}%
3249     \let\bbl@ld@calendar\@empty
3250     \let\bbl@ld@variant\@empty
3251     \let\bbl@ld@convert\relax
3252     \def\bbl@tempb##1=##2\@@{\@namedef{bbl@ld@##1}{##2}}%
3253     \bbl@foreach\bbl@tempe{\bbl@tempb##1\@@}%
3254     \bbl@replace\bbl@ld@calendar{gregorian}{}%
3255     \ifx\bbl@ld@calendar\@empty\else
3256       \ifx\bbl@ld@convert\relax\else
3257         \babelcalendar[\bbl@they-\bbl@them-\bbl@thed]%
3258           {\bbl@ld@calendar}\bbl@they\bbl@them\bbl@thed
```

```
3259      \fi
3260     \fi
3261    \@nameuse{bbl@precalendar}% Remove, eg, +, -civil (-ca-islamic)
3262    \edef\bbl@calendar{% Used in \month..., too
3263      \bbl@ld@calendar
3264      \ifx\bbl@ld@variant\@empty\else
3265        .\bbl@ld@variant
3266      \fi}%
3267    \bbl@cased
3268      {\@nameuse{bbl@date@\languagename @\bbl@calendar}%
3269          \bbl@they\bbl@them\bbl@thed}%
3270   \endgroup}
3271 % eg: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
3272 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
3273   \bbl@trim@def\bbl@tempa{#1.#2}%
3274   \bbl@ifsamestring{\bbl@tempa}{months.wide}%         to savedate
3275     {\bbl@trim@def\bbl@tempa{#3}%
3276      \bbl@trim\toks@{#5}%
3277      \@temptokena\expandafter{\bbl@savedate}%
3278      \bbl@exp{%   Reverse order - in ini last wins
3279        \def\\\bbl@savedate{%
3280          \\\SetString\<month\romannumeral\bbl@tempa#6name>{\the\toks@}%
3281          \the\@temptokena}}}%
3282     {\bbl@ifsamestring{\bbl@tempa}{date.long}%       defined now
3283       {\lowercase{\def\bbl@tempb{#6}}%
3284        \bbl@trim@def\bbl@toreplace{#5}%
3285        \bbl@TG@@date
3286        \global\bbl@csarg\let{date@\languagename @\bbl@tempb}\bbl@toreplace
3287        \ifx\bbl@savetoday\@empty
3288          \bbl@exp{% TODO. Move to a better place.
3289            \\\AfterBabelCommands{%
3290              \def\<\languagename date>{\\\protect\<\languagename date >}%
3291              \\\newcommand\<\languagename date >[4][]{%
3292                \\\bbl@usedategrouptrue
3293                \<bbl@ensure@\languagename>{%
3294                  \\\localedate[####1]{####2}{####3}{####4}}}}%
3295          \def\\\bbl@savetoday{%
3296            \\\SetString\\\today{%
3297              \<\languagename date>[convert]%
3298                {\\\the\year}{\\\the\month}{\\\the\day}}}}%
3299        \fi}%
3300        {}}}
```

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).

```
3301 \let\bbl@calendar\@empty
3302 \newcommand\babelcalendar[2][\the\year-\the\month-\the\day]{%
3303   \@nameuse{bbl@ca@#2}#1\@@}
3304 \newcommand\BabelDateSpace{\nobreakspace}
3305 \newcommand\BabelDateDot{.\@}  % TODO. \let instead of repeating
3306 \newcommand\BabelDated[1]{{\number#1}}
3307 \newcommand\BabelDatedd[1]{{\ifnum#1<10 0\fi\number#1}}
3308 \newcommand\BabelDateM[1]{{\number#1}}
3309 \newcommand\BabelDateMM[1]{{\ifnum#1<10 0\fi\number#1}}
3310 \newcommand\BabelDateMMMM[1]{{%
3311   \csname month\romannumeral#1\bbl@calendar name\endcsname}}%
3312 \newcommand\BabelDatey[1]{{\number#1}}%
3313 \newcommand\BabelDateyy[1]{{%
3314   \ifnum#1<10 0\number#1 %
3315   \else\ifnum#1<100 \number#1 %
```

```
3316    \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3317    \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
3318    \else
3319      \bbl@error
3320        {Currently two-digit years are restricted to the\\
3321         range 0-9999.}%
3322        {There is little you can do. Sorry.}%
3323    \fi\fi\fi\fi}}
3324 \newcommand\BabelDateyyyy[1]{{\number#1}} % TODO - add leading 0
3325 \def\bbl@replace@finish@iii#1{%
3326    \bbl@exp{\def\\#1####1####2####3{\the\toks@}}}
3327 \def\bbl@TG@@date{%
3328    \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace{}}%
3329    \bbl@replace\bbl@toreplace{[.]}{\BabelDateDot{}}%
3330    \bbl@replace\bbl@toreplace{[d]}{\BabelDated{####3}}%
3331    \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{####3}}%
3332    \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{####2}}%
3333    \bbl@replace\bbl@toreplace{[MM]}{\BabelDateMM{####2}}%
3334    \bbl@replace\bbl@toreplace{[MMMM]}{\BabelDateMMMM{####2}}%
3335    \bbl@replace\bbl@toreplace{[y]}{\BabelDatey{####1}}%
3336    \bbl@replace\bbl@toreplace{[yy]}{\BabelDateyy{####1}}%
3337    \bbl@replace\bbl@toreplace{[yyyy]}{\BabelDateyyyy{####1}}%
3338    \bbl@replace\bbl@toreplace{[y|}{\bbl@datecntr[####1|}%
3339    \bbl@replace\bbl@toreplace{[m|}{\bbl@datecntr[####2|}%
3340    \bbl@replace\bbl@toreplace{[d|}{\bbl@datecntr[####3|}%
3341    \bbl@replace@finish@iii\bbl@toreplace}
3342 \def\bbl@datecntr{\expandafter\bbl@xdatecntr\expandafter}
3343 \def\bbl@xdatecntr[#1|#2]{\localenumeral{#2}{#1}}
```

Transforms.

```
3344 \let\bbl@release@transforms\@empty
3345 \@namedef{bbl@inikv@transforms.prehyphenation}{%
3346    \bbl@transforms\babelprehyphenation}
3347 \@namedef{bbl@inikv@transforms.posthyphenation}{%
3348    \bbl@transforms\babelposthyphenation}
3349 \def\bbl@transforms@aux#1#2#3#4,#5\relax{%
3350    #1[#2]{#3}{#4}{#5}}
3351 \begingroup %  A hack. TODO. Don't require an specific order
3352    \catcode`\%=12
3353    \catcode`\&=14
3354    \gdef\bbl@transforms#1#2#3{&%
3355      \ifx\bbl@KVP@transforms\@nnil\else
3356        \directlua{
3357          local str = [==[#2]==]
3358          str = str:gsub('%.%d+%.%d+$', '')
3359          tex.print([[\def\string\babeltempa{]] .. str .. [[}]])
3360        }&%
3361      \bbl@xin@{,\babeltempa,}{,\bbl@KVP@transforms,}&%
3362      \ifin@
3363        \in@{.0$}{#2$}&%
3364        \ifin@
3365          \directlua{
3366            local str = string.match([[\bbl@KVP@transforms]],
3367                       '%(([^%(]-)%)[^%)]-\babeltempa')
3368            if str == nil then
3369              tex.print([[\def\string\babeltempb{}]])
3370            else
3371              tex.print([[\def\string\babeltempb{,attribute=]] .. str .. [[}]])
3372            end
3373          }
3374        \toks@{#3}&%
3375        \bbl@exp{&%
3376          \\\g@addto@macro\\\bbl@release@transforms{&%
```

```
3377              \relax  &% Closes previous \bbl@transforms@aux
3378              \\\bbl@transforms@aux
3379                \\#1{label=\babeltempa\babeltempb}{\languagename}{\the\toks@}}}&%
3380          \else
3381            \g@addto@macro\bbl@release@transforms{, {#3}}&%
3382          \fi
3383        \fi
3384      \fi}
3385 \endgroup
```

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

```
3386 \def\bbl@provide@lsys#1{%
3387    \bbl@ifunset{bbl@lname@#1}%
3388      {\bbl@load@info{#1}}%
3389      {}%
3390    \bbl@csarg\let{lsys@#1}\@empty
3391    \bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}{}%
3392    \bbl@ifunset{bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{}%
3393    \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
3394    \bbl@ifunset{bbl@lname@#1}{}%
3395      {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
3396    \ifcase\bbl@engine\or\or
3397      \bbl@ifunset{bbl@prehc@#1}{}%
3398        {\bbl@exp{\\\bbl@ifblank{\bbl@cs{prehc@#1}}}%
3399          {}%
3400          {\ifx\bbl@xenohyph\@undefined
3401              \global\let\bbl@xenohyph\bbl@xenohyph@d
3402              \ifx\AtBeginDocument\@notprerr
3403                \expandafter\@secondoftwo  % to execute right now
3404              \fi
3405              \AtBeginDocument{%
3406                \bbl@patchfont{\bbl@xenohyph}%
3407                \expandafter\selectlanguage\expandafter{\languagename}}%
3408          \fi}}%
3409    \fi
3410    \bbl@csarg\bbl@toglobal{lsys@#1}}
3411 \def\bbl@xenohyph@d{%
3412    \bbl@ifset{bbl@prehc@\languagename}%
3413      {\ifnum\hyphenchar\font=\defaulthyphenchar
3414          \iffontchar\font\bbl@cl{prehc}\relax
3415            \hyphenchar\font\bbl@cl{prehc}\relax
3416          \else\iffontchar\font"200B
3417            \hyphenchar\font"200B
3418          \else
3419            \bbl@warning
3420              {Neither 0 nor ZERO WIDTH SPACE are available\\%
3421               in the current font, and therefore the hyphen\\%
3422               will be printed. Try changing the fontspec's\\%
3423               'HyphenChar' to another value, but be aware\\%
3424               this setting is not safe (see the manual)}%
3425            \hyphenchar\font\defaulthyphenchar
3426          \fi\fi
3427        \fi}%
3428      {\hyphenchar\font\defaulthyphenchar}}
3429 % \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).

```
3430 \def\bbl@load@info#1{%
3431    \def\BabelBeforeIni##1##2{%
3432      \begingroup
```

```
3433        \bbl@read@ini{##1}0%
3434          \endinput          % babel- .tex may contain onlypreamble's
3435        \endgroup}%              boxed, to avoid extra spaces:
3436    {\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.

```
3437 \def\bbl@setdigits#1#2#3#4#5{%
3438   \bbl@exp{%
3439     \def\<\languagename digits>####1{%        ie, \langdigits
3440       \<bbl@digits@\languagename>####1\\\@nil}%
3441     \let\<bbl@cntr@digits@\languagename>\<\languagename digits>%
3442     \def\<\languagename counter>####1{%      ie, \langcounter
3443       \\\expandafter\<bbl@counter@\languagename>%
3444       \\\csname c@####1\endcsname}%
3445     \def\<bbl@counter@\languagename>####1{% ie, \bbl@counter@lang
3446       \\\expandafter\<bbl@digits@\languagename>%
3447       \\\number####1\\\@nil}}%
3448   \def\bbl@tempa##1##2##3##4##5{%
3449     \bbl@exp{%    Wow, quite a lot of hashes! :-(
3450       \def\<bbl@digits@\languagename>########1{%
3451        \\\ifx########1\\\@nil              % ie, \bbl@digits@lang
3452        \\\else
3453          \\\ifx0########1#1%
3454          \\\else\\\ifx1########1#2%
3455          \\\else\\\ifx2########1#3%
3456          \\\else\\\ifx3########1#4%
3457          \\\else\\\ifx4########1#5%
3458          \\\else\\\ifx5########1##1%
3459          \\\else\\\ifx6########1##2%
3460          \\\else\\\ifx7########1##3%
3461          \\\else\\\ifx8########1##4%
3462          \\\else\\\ifx9########1##5%
3463          \\\else########1%
3464          \\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi
3465          \\\expandafter\<bbl@digits@\languagename>%
3466        \\\fi}}}%
3467   \bbl@tempa}
```

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

```
3468 \def\bbl@buildifcase#1 {% Returns \bbl@tempa, requires \toks@={}
3469   \ifx\\#1%                % \\ before, in case #1 is multiletter
3470     \bbl@exp{%
3471       \def\\bbl@tempa####1{%
3472         \<ifcase>####1\space\the\toks@\<else>\\\@ctrerr\<fi>}}%
3473   \else
3474     \toks@\expandafter{\the\toks@\or #1}%
3475     \expandafter\bbl@buildifcase
3476   \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).

```
3477 \newcommand\localenumeral[2]{\bbl@cs{cntr@#1@\languagename}{#2}}
3478 \def\bbl@localecntr#1#2{\localenumeral{#2}{#1}}
3479 \newcommand\localecounter[2]{%
3480   \expandafter\bbl@localecntr
3481   \expandafter{\number\csname c@#2\endcsname}{#1}}
3482 \def\bbl@alphnumeral#1#2{%
3483   \expandafter\bbl@alphnumeral@i\number#2 76543210\@@{#1}}
3484 \def\bbl@alphnumeral@i#1#2#3#4#5#6#7#8\@@#9{%
```

```
3485    \ifcase\@car#8\@nil\or    % Currenty <10000, but prepared for bigger
3486      \bbl@alphnumeral@ii{#9}000000#1\or
3487      \bbl@alphnumeral@ii{#9}00000#1#2\or
3488      \bbl@alphnumeral@ii{#9}0000#1#2#3\or
3489      \bbl@alphnumeral@ii{#9}000#1#2#3#4\else
3490      \bbl@alphnum@invalid{>9999}%
3491    \fi}
3492 \def\bbl@alphnumeral@ii#1#2#3#4#5#6#7#8{%
3493    \bbl@ifunset{bbl@cntr@#1.F.\number#5#6#7#8@\languagename}%
3494      {\bbl@cs{cntr@#1.4@\languagename}#5%
3495       \bbl@cs{cntr@#1.3@\languagename}#6%
3496       \bbl@cs{cntr@#1.2@\languagename}#7%
3497       \bbl@cs{cntr@#1.1@\languagename}#8%
3498       \ifnum#6#7#8>\z@ % TODO. An ad hoc rule for Greek. Ugly.
3499         \bbl@ifunset{bbl@cntr@#1.S.321@\languagename}{}%
3500           {\bbl@cs{cntr@#1.S.321@\languagename}}%
3501       \fi}%
3502      {\bbl@cs{cntr@#1.F.\number#5#6#7#8@\languagename}}}
3503 \def\bbl@alphnum@invalid#1{%
3504    \bbl@error{Alphabetic numeral too large (#1)}%
3505      {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.

```
3506 \def\bbl@localeinfo#1#2{%
3507    \bbl@ifunset{bbl@info@#2}{#1}%
3508      {\bbl@ifunset{bbl@\csname bbl@info@#2\endcsname @\languagename}{#1}%
3509        {\bbl@cs{\csname bbl@info@#2\endcsname @\languagename}}}}
3510 \newcommand\localeinfo[1]{%
3511    \ifx*#1\@empty    % TODO. A bit hackish to make it expandable.
3512      \bbl@afterelse\bbl@localeinfo{}%
3513    \else
3514      \bbl@localeinfo
3515        {\bbl@error{I've found no info for the current locale.\\%
3516                    The corresponding ini file has not been loaded\\%
3517                    Perhaps it doesn't exist}%
3518                   {See the manual for details.}}%
3519        {#1}%
3520    \fi}
3521 % \@namedef{bbl@info@name.locale}{lcname}
3522 \@namedef{bbl@info@tag.ini}{lini}
3523 \@namedef{bbl@info@name.english}{elname}
3524 \@namedef{bbl@info@name.opentype}{lname}
3525 \@namedef{bbl@info@tag.bcp47}{tbcp}
3526 \@namedef{bbl@info@language.tag.bcp47}{lbcp}
3527 \@namedef{bbl@info@tag.opentype}{lotf}
3528 \@namedef{bbl@info@script.name}{esname}
3529 \@namedef{bbl@info@script.name.opentype}{sname}
3530 \@namedef{bbl@info@script.tag.bcp47}{sbcp}
3531 \@namedef{bbl@info@script.tag.opentype}{sotf}
3532 \@namedef{bbl@info@region.tag.bcp47}{rbcp}
3533 \@namedef{bbl@info@variant.tag.bcp47}{vbcp}
3534 % Extensions are dealt with in a special way
3535 % Now, an internal \LaTeX{} macro:
3536 \providecommand\BCPdata[1]{\localeinfo*{#1.tag.bcp47}}
```

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

```
3537 ⟨⟨*More package options⟩⟩ ≡
3538 \DeclareOption{ensureinfo=off}{}
3539 ⟨⟨/More package options⟩⟩
3540 %
3541 \let\bbl@ensureinfo\@gobble
3542 \newcommand\BabelEnsureInfo{%
3543    \ifx\InputIfFileExists\@undefined\else
```

```
3544     \def\bbl@ensureinfo##1{%
3545       \bbl@ifunset{bbl@lname@##1}{\bbl@load@info{##1}}{}}%
3546   \fi
3547   \bbl@foreach\bbl@loaded{{%
3548     \def\languagename{##1}%
3549     \bbl@ensureinfo{##1}}}}
3550 \@ifpackagewith{babel}{ensureinfo=off}{}%
3551   {\AtEndOfPackage{% Test for plain.
3552     \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`.

```
3553 \newcommand\getlocaleproperty{%
3554   \@ifstar\bbl@getproperty@s\bbl@getproperty@x}
3555 \def\bbl@getproperty@s#1#2#3{%
3556   \let#1\relax
3557   \def\bbl@elt##1##2##3{%
3558     \bbl@ifsamestring{##1/##2}{#3}%
3559       {\providecommand#1{##3}%
3560        \def\bbl@elt####1####2####3{}}%
3561       {}}%
3562   \bbl@cs{inidata@#2}}%
3563 \def\bbl@getproperty@x#1#2#3{%
3564   \bbl@getproperty@s{#1}{#2}{#3}%
3565   \ifx#1\relax
3566     \bbl@error
3567       {Unknown key for locale '#2':\\%
3568        #3\\%
3569        \string#1 will be set to \relax}%
3570       {Perhaps you misspelled it.}%
3571   \fi}
3572 \let\bbl@ini@loaded\@empty
3573 \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.

```
3574 \newcommand\babeladjust[1]{%  TODO. Error handling.
3575   \bbl@forkv{#1}{%
3576     \bbl@ifunset{bbl@ADJ@##1@##2}%
3577       {\bbl@cs{ADJ@##1}{##2}}%
3578       {\bbl@cs{ADJ@##1@##2}}}}
3579 %
3580 \def\bbl@adjust@lua#1#2{%
3581   \ifvmode
3582     \ifnum\currentgrouplevel=\z@
3583       \directlua{ Babel.#2 }%
3584       \expandafter\expandafter\expandafter\@gobble
3585     \fi
3586   \fi
3587   {\bbl@error    % The error is gobbled if everything went ok.
3588      {Currently, #1 related features can be adjusted only\\%
3589       in the main vertical list.}%
3590      {Maybe things change in the future, but this is what it is.}}}
3591 \@namedef{bbl@ADJ@bidi.mirroring@on}{%
3592   \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
3593 \@namedef{bbl@ADJ@bidi.mirroring@off}{%
3594   \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
3595 \@namedef{bbl@ADJ@bidi.text@on}{%
3596   \bbl@adjust@lua{bidi}{bidi_enabled=true}}
3597 \@namedef{bbl@ADJ@bidi.text@off}{%
```

```
3598    \bbl@adjust@lua{bidi}{bidi_enabled=false}}
3599  \@namedef{bbl@ADJ@bidi.mapdigits@on}{%
3600    \bbl@adjust@lua{bidi}{digits_mapped=true}}
3601  \@namedef{bbl@ADJ@bidi.mapdigits@off}{%
3602    \bbl@adjust@lua{bidi}{digits_mapped=false}}
3603  %
3604  \@namedef{bbl@ADJ@linebreak.sea@on}{%
3605    \bbl@adjust@lua{linebreak}{sea_enabled=true}}
3606  \@namedef{bbl@ADJ@linebreak.sea@off}{%
3607    \bbl@adjust@lua{linebreak}{sea_enabled=false}}
3608  \@namedef{bbl@ADJ@linebreak.cjk@on}{%
3609    \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
3610  \@namedef{bbl@ADJ@linebreak.cjk@off}{%
3611    \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
3612  \@namedef{bbl@ADJ@justify.arabic@on}{%
3613    \bbl@adjust@lua{linebreak}{arabic.justify_enabled=true}}
3614  \@namedef{bbl@ADJ@justify.arabic@off}{%
3615    \bbl@adjust@lua{linebreak}{arabic.justify_enabled=false}}
3616  %
3617  \def\bbl@adjust@layout#1{%
3618    \ifvmode
3619      #1%
3620      \expandafter\@gobble
3621    \fi
3622    {\bbl@error    % The error is gobbled if everything went ok.
3623      {Currently, layout related features can be adjusted only\\%
3624        in vertical mode.}%
3625      {Maybe things change in the future, but this is what it is.}}}
3626  \@namedef{bbl@ADJ@layout.tabular@on}{%
3627    \bbl@adjust@layout{\let\@tabular\bbl@NL@@tabular}}
3628  \@namedef{bbl@ADJ@layout.tabular@off}{%
3629    \bbl@adjust@layout{\let\@tabular\bbl@OL@@tabular}}
3630  \@namedef{bbl@ADJ@layout.lists@on}{%
3631    \bbl@adjust@layout{\let\list\bbl@NL@list}}
3632  \@namedef{bbl@ADJ@layout.lists@off}{%
3633    \bbl@adjust@layout{\let\list\bbl@OL@list}}
3634  \@namedef{bbl@ADJ@hyphenation.extra@on}{%
3635    \bbl@activateposthyphen}
3636  %
3637  \@namedef{bbl@ADJ@autoload.bcp47@on}{%
3638    \bbl@bcpallowedtrue}
3639  \@namedef{bbl@ADJ@autoload.bcp47@off}{%
3640    \bbl@bcpallowedfalse}
3641  \@namedef{bbl@ADJ@autoload.bcp47.prefix}#1{%
3642    \def\bbl@bcp@prefix{#1}}
3643  \def\bbl@bcp@prefix{bcp47-}
3644  \@namedef{bbl@ADJ@autoload.options}#1{%
3645    \def\bbl@autoload@options{#1}}
3646  \let\bbl@autoload@bcpoptions\@empty
3647  \@namedef{bbl@ADJ@autoload.bcp47.options}#1{%
3648    \def\bbl@autoload@bcpoptions{#1}}
3649  \newif\ifbbl@bcptoname
3650  \@namedef{bbl@ADJ@bcp47.toname@on}{%
3651    \bbl@bcptonametrue
3652    \BabelEnsureInfo}
3653  \@namedef{bbl@ADJ@bcp47.toname@off}{%
3654    \bbl@bcptonamefalse}
3655  \@namedef{bbl@ADJ@prehyphenation.disable@nohyphenation}{%
3656    \directlua{ Babel.ignore_pre_char = function(node)
3657        return (node.lang == \the\csname l@nohyphenation\endcsname)
3658      end }}
3659  \@namedef{bbl@ADJ@prehyphenation.disable@off}{%
3660    \directlua{ Babel.ignore_pre_char = function(node)
```

```
3661        return false
3662      end }}
3663 \@namedef{bbl@ADJ@select.write@shift}{%
3664    \let\bbl@restorelastskip\relax
3665    \def\bbl@savelastskip{%
3666      \let\bbl@restorelastskip\relax
3667      \ifvmode
3668        \ifdim\lastskip=\z@
3669          \let\bbl@restorelastskip\nobreak
3670        \else
3671          \bbl@exp{%
3672            \def\\\bbl@restorelastskip{%
3673              \skip@=\the\lastskip
3674              \\\nobreak \vskip-\skip@ \vskip\skip@}}%
3675        \fi
3676      \fi}}
3677 \@namedef{bbl@ADJ@select.write@keep}{%
3678    \let\bbl@restorelastskip\relax
3679    \let\bbl@savelastskip\relax}
3680 \@namedef{bbl@ADJ@select.write@omit}{%
3681    \let\bbl@restorelastskip\relax
3682    \def\bbl@savelastskip##1\bbl@restorelastskip{}}
```

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

```
3683 \ifx\directlua\@undefined\else
3684    \ifx\bbl@luapatterns\@undefined
3685      \input luababel.def
3686    \fi
3687 \fi
```

Continue with LaTeX.

```
3688 ⟨/package | core⟩
3689 ⟨∗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.

```
3690 ⟨⟨∗More package options⟩⟩ ≡
3691 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
3692 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
3693 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
3694 \DeclareOption{safe=refbib}{\def\bbl@opt@safe{BR}}
3695 \DeclareOption{safe=bibref}{\def\bbl@opt@safe{BR}}
3696 ⟨⟨/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.

```
3697 \bbl@trace{Cross referencing macros}
3698 \ifx\bbl@opt@safe\@empty\else % ie, if 'ref' and/or 'bib'
3699    \def\@newl@bel#1#2#3{%
3700      {\@safe@activestrue
3701        \bbl@ifunset{#1@#2}%
3702          \relax
```

```
3703        {\gdef\@multiplelabels{%
3704            \@latex@warning@no@line{There were multiply-defined labels}}%
3705          \@latex@warning@no@line{Label `#2' multiply defined}}%
3706      \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.

```
3707  \CheckCommand*\@testdef[3]{%
3708    \def\reserved@a{#3}%
3709    \expandafter\ifx\csname#1@#2\endcsname\reserved@a
3710    \else
3711      \@tempswatrue
3712    \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.

```
3713  \def\@testdef#1#2#3{%  TODO. With @samestring?
3714    \@safe@activestrue
3715    \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
3716    \def\bbl@tempb{#3}%
3717    \@safe@activesfalse
3718    \ifx\bbl@tempa\relax
3719    \else
3720      \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
3721    \fi
3722    \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
3723    \ifx\bbl@tempa\bbl@tempb
3724    \else
3725      \@tempswatrue
3726    \fi}
3727 \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.

```
3728 \bbl@xin@{R}\bbl@opt@safe
3729 \ifin@
3730  \edef\bbl@tempc{\expandafter\string\csname ref code\endcsname}%
3731  \bbl@xin@{\expandafter\strip@prefix\meaning\bbl@tempc}%
3732    {\expandafter\strip@prefix\meaning\ref}%
3733  \ifin@
3734    \bbl@redefine\@kernel@ref#1{%
3735      \@safe@activestrue\org@@kernel@ref{#1}\@safe@activesfalse}
3736    \bbl@redefine\@kernel@pageref#1{%
3737      \@safe@activestrue\org@@kernel@pageref{#1}\@safe@activesfalse}
3738    \bbl@redefine\@kernel@sref#1{%
3739      \@safe@activestrue\org@@kernel@sref{#1}\@safe@activesfalse}
3740    \bbl@redefine\@kernel@spageref#1{%
3741      \@safe@activestrue\org@@kernel@spageref{#1}\@safe@activesfalse}
3742  \else
3743    \bbl@redefinerobust\ref#1{%
3744      \@safe@activestrue\org@ref{#1}\@safe@activesfalse}
3745    \bbl@redefinerobust\pageref#1{%
3746      \@safe@activestrue\org@pageref{#1}\@safe@activesfalse}
3747  \fi
3748 \else
3749  \let\org@ref\ref
3750  \let\org@pageref\pageref
3751 \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.

```
3752 \bbl@xin@{B}\bbl@opt@safe
3753 \ifin@
3754   \bbl@redefine\@citex[#1]#2{%
3755     \@safe@activestrue\edef\@tempa{#2}\@safe@activesfalse
3756     \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.

```
3757 \AtBeginDocument{%
3758   \@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.)

```
3759     \def\@citex[#1][#2]#3{%
3760       \@safe@activestrue\edef\@tempa{#3}\@safe@activesfalse
3761       \org@@citex[#1][#2]{\@tempa}}%
3762   }{}}
```

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

```
3763 \AtBeginDocument{%
3764   \@ifpackageloaded{cite}{%
3765     \def\@citex[#1]#2{%
3766       \@safe@activestrue\org@@citex[#1]{#2}\@safe@activesfalse}%
3767   }{}}
```

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

```
3768 \bbl@redefine\nocite#1{%
3769   \@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.

```
3770 \bbl@redefine\bibcite{%
3771   \bbl@cite@choice
3772   \bibcite}
```

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

```
3773 \def\bbl@bibcite#1#2{%
3774   \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.

```
3775 \def\bbl@cite@choice{%
3776   \global\let\bibcite\bbl@bibcite
3777   \@ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{}%
3778   \@ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{}%
3779   \global\let\bbl@cite@choice\relax}
```

138

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.

```
3780    \AtBeginDocument{\bbl@cite@choice}
```

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

```
3781    \bbl@redefine\@bibitem#1{%
3782      \@safe@activestrue\org@@bibitem{#1}\@safe@activesfalse}
3783  \else
3784    \let\org@nocite\nocite
3785    \let\org@@citex\@citex
3786    \let\org@bibcite\bibcite
3787    \let\org@@bibitem\@bibitem
3788  \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.

```
3789  \bbl@trace{Marks}
3790  \IfBabelLayout{sectioning}
3791    {\ifx\bbl@opt@headfoot\@nnil
3792       \g@addto@macro\@resetactivechars{%
3793         \set@typeset@protect
3794         \expandafter\select@language@x\expandafter{\bbl@main@language}%
3795         \let\protect\noexpand
3796         \ifcase\bbl@bidimode\else % Only with bidi. See also above
3797           \edef\thepage{%
3798             \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
3799         \fi}%
3800      \fi}
3801    {\ifbbl@single\else
3802       \bbl@ifunset{markright }\bbl@redefine\bbl@redefinerobust
3803       \markright#1{%
3804         \bbl@ifblank{#1}%
3805           {\org@markright{}}%
3806           {\toks@{#1}%
3807            \bbl@exp{%
3808              \\\org@markright{\\\protect\\\foreignlanguage{\languagename}%
3809                {\\\protect\\\bbl@restore@actives\the\toks@}}}}}%
```

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

```
3810      \ifx\@mkboth\markboth
3811        \def\bbl@tempc{\let\@mkboth\markboth}
3812      \else
3813        \def\bbl@tempc{}
3814      \fi
3815      \bbl@ifunset{markboth }\bbl@redefine\bbl@redefinerobust
3816      \markboth#1#2{%
3817        \protected@edef\bbl@tempb##1{%
3818          \protect\foreignlanguage
3819          {\languagename}{\protect\bbl@restore@actives##1}}%
3820        \bbl@ifblank{#1}%
3821          {\toks@{}}%
```

```
3822            {\toks@\expandafter{\bbl@tempb{#1}}}%
3823          \bbl@ifblank{#2}%
3824            {\@temptokena{}}%
3825            {\@temptokena\expandafter{\bbl@tempb{#2}}}%
3826          \bbl@exp{\\\org@markboth{\the\toks@}{\the\@temptokena}}}
3827          \bbl@tempc
3828        \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.

```
3829 \bbl@trace{Preventing clashes with other packages}
3830 \ifx\org@ref\@undefined\else
3831   \bbl@xin@{R}\bbl@opt@safe
3832   \ifin@
3833     \AtBeginDocument{%
3834       \@ifpackageloaded{ifthen}{%
3835         \bbl@redefine@long\ifthenelse#1#2#3{%
3836           \let\bbl@temp@pref\pageref
3837           \let\pageref\org@pageref
3838           \let\bbl@temp@ref\ref
3839           \let\ref\org@ref
3840           \@safe@activestrue
3841           \org@ifthenelse{#1}%
3842             {\let\pageref\bbl@temp@pref
3843              \let\ref\bbl@temp@ref
3844              \@safe@activesfalse
3845              #2}%
3846             {\let\pageref\bbl@temp@pref
3847              \let\ref\bbl@temp@ref
3848              \@safe@activesfalse
3849              #3}%
3850         }%
3851       }{}%
3852     }
3853 \fi
```

8.3.2 `varioref`

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

```
3854     \AtBeginDocument{%
3855       \@ifpackageloaded{varioref}{%
3856         \bbl@redefine\@@vpageref#1[#2]#3{%
3857           \@safe@activestrue
```

```
3858        \org@@@vpageref{#1}[#2]{#3}%
3859        \@safe@activesfalse}%
3860      \bbl@redefine\vrefpagenum#1#2{%
3861        \@safe@activestrue
3862        \org@vrefpagenum{#1}{#2}%
3863        \@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.

```
3864      \expandafter\def\csname Ref \endcsname#1{%
3865        \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
3866      }{}%
3867    }
3868 \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.

```
3869 \AtEndOfPackage{%
3870  \AtBeginDocument{%
3871    \@ifpackageloaded{hhline}%
3872      {\expandafter\ifx\csname normal@char\string:\endcsname\relax
3873        \else
3874          \makeatletter
3875          \def\@currname{hhline}\input{hhline.sty}\makeatother
3876        \fi}%
3877      {}}}
```

\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.

```
3878 \def\substitutefontfamily#1#2#3{%
3879  \lowercase{\immediate\openout15=#1#2.fd\relax}%
3880  \immediate\write15{%
3881    \string\ProvidesFile{#1#2.fd}%
3882    [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
3883     \space generated font description file]^^J
3884    \string\DeclareFontFamily{#1}{#2}{}^^J
3885    \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{}^^J
3886    \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{}^^J
3887    \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{}^^J
3888    \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{}^^J
3889    \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{}^^J
3890    \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{}^^J
3891    \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{}^^J
3892    \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{}^^J
3893    }%
3894  \closeout15
3895  }
3896 \@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.

```
3897 \bbl@trace{Encoding and fonts}
3898 \newcommand\BabelNonASCII{LGR,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU}
3899 \newcommand\BabelNonText{TS1,T3,TS3}
3900 \let\org@TeX\TeX
3901 \let\org@LaTeX\LaTeX
3902 \let\ensureascii\@firstofone
3903 \AtBeginDocument{%
3904   \def\@elt#1{,#1,}%
3905   \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3906   \let\@elt\relax
3907   \let\bbl@tempb\@empty
3908   \def\bbl@tempc{OT1}%
3909   \bbl@foreach\BabelNonASCII{% LGR loaded in a non-standard way
3910     \bbl@ifunset{T@#1}{}{\def\bbl@tempb{#1}}}%
3911   \bbl@foreach\bbl@tempa{%
3912     \bbl@xin@{#1}{\BabelNonASCII}%
3913     \ifin@
3914       \def\bbl@tempb{#1}% Store last non-ascii
3915     \else\bbl@xin@{#1}{\BabelNonText}% Pass
3916       \ifin@\else
3917         \def\bbl@tempc{#1}% Store last ascii
3918       \fi
3919     \fi}%
3920   \ifx\bbl@tempb\@empty\else
3921     \bbl@xin@{,\cf@encoding,}{,\BabelNonASCII,\BabelNonText,}%
3922     \ifin@\else
3923       \edef\bbl@tempc{\cf@encoding}% The default if ascii wins
3924     \fi
3925     \edef\ensureascii#1{%
3926       {\noexpand\fontencoding{\bbl@tempc}\noexpand\selectfont#1}}%
3927     \DeclareTextCommandDefault{\TeX}{\ensureascii{\org@TeX}}%
3928     \DeclareTextCommandDefault{\LaTeX}{\ensureascii{\org@LaTeX}}%
3929   \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.

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.

```
3930 \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.

```
3931 \AtBeginDocument{%
3932   \@ifpackageloaded{fontspec}%
3933     {\xdef\latinencoding{%
3934       \ifx\UTFencname\@undefined
3935         EU\ifcase\bbl@engine\or2\or1\fi
3936       \else
3937         \UTFencname
3938       \fi}}%
3939     {\gdef\latinencoding{OT1}%
3940      \ifx\cf@encoding\bbl@t@one
3941        \xdef\latinencoding{\bbl@t@one}%
3942      \else
3943        \def\@elt#1{,#1,}%
```

```
3944      \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3945      \let\@elt\relax
3946      \bbl@xin@{,T1,}\bbl@tempa
3947      \ifin@
3948        \xdef\latinencoding{\bbl@t@one}%
3949      \fi
3950    \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.

```
3951 \DeclareRobustCommand{\latintext}{%
3952   \fontencoding{\latinencoding}\selectfont
3953   \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.

```
3954 \ifx\@undefined\DeclareTextFontCommand
3955   \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
3956 \else
3957   \DeclareTextFontCommand{\textlatin}{\latintext}
3958 \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).

```
3959 \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.

```
3960 \bbl@trace{Loading basic (internal) bidi support}
3961 \ifodd\bbl@engine
3962 \else % TODO. Move to txtbabel
3963   \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
3964     \bbl@error
3965       {The bidi method 'basic' is available only in\\%
3966        luatex. I'll continue with 'bidi=default', so\\%
3967        expect wrong results}%
3968       {See the manual for further details.}%
3969     \let\bbl@beforeforeign\leavevmode
3970     \AtEndOfPackage{%
3971       \EnableBabelHook{babel-bidi}%
3972       \bbl@xebidipar}
```

```
3973  \fi\fi
3974  \def\bbl@loadxebidi#1{%
3975    \ifx\RTLfootnotetext\@undefined
3976      \AtEndOfPackage{%
3977        \EnableBabelHook{babel-bidi}%
3978        \bbl@loadfontspec % bidi needs fontspec
3979        \usepackage#1{bidi}}%
3980    \fi}
3981  \ifnum\bbl@bidimode>200
3982    \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
3983      \bbl@tentative{bidi=bidi}
3984      \bbl@loadxebidi{}
3985    \or
3986      \bbl@loadxebidi{[rldocument]}
3987    \or
3988      \bbl@loadxebidi{}
3989    \fi
3990  \fi
3991 \fi
3992 % TODO? Separate:
3993 \ifnum\bbl@bidimode=\@ne
3994   \let\bbl@beforeforeign\leavevmode
3995   \ifodd\bbl@engine
3996     \newattribute\bbl@attr@dir
3997     \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
3998     \bbl@exp{\output{\bodydir\pagedir\the\output}}
3999   \fi
4000   \AtEndOfPackage{%
4001     \EnableBabelHook{babel-bidi}%
4002     \ifodd\bbl@engine\else
4003       \bbl@xebidipar
4004     \fi}
4005 \fi
```

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

```
4006 \bbl@trace{Macros to switch the text direction}
4007 \def\bbl@alscripts{,Arabic,Syriac,Thaana,}
4008 \def\bbl@rscripts{% TODO. Base on codes ??
4009   ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
4010   Old Hungarian,Old Hungarian,Lydian,Mandaean,Manichaean,%
4011   Manichaean,Meroitic Cursive,Meroitic,Old North Arabian,%
4012   Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
4013   Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
4014   Old South Arabian,}%
4015 \def\bbl@provide@dirs#1{%
4016   \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
4017   \ifin@
4018     \global\bbl@csarg\chardef{wdir@#1}\@ne
4019     \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
4020     \ifin@
4021       \global\bbl@csarg\chardef{wdir@#1}\tw@  % useless in xetex
4022     \fi
4023   \else
4024     \global\bbl@csarg\chardef{wdir@#1}\z@
4025   \fi
4026   \ifodd\bbl@engine
4027     \bbl@csarg\ifcase{wdir@#1}%
4028       \directlua{ Babel.locale_props[\the\localeid].textdir = 'l' }%
4029     \or
4030       \directlua{ Babel.locale_props[\the\localeid].textdir = 'r' }%
4031     \or
4032       \directlua{ Babel.locale_props[\the\localeid].textdir = 'al' }%
```

```
4033      \fi
4034   \fi}
4035 \def\bbl@switchdir{%
4036   \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
4037   \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
4038   \bbl@exp{\\\bbl@setdirs\bbl@cl{wdir}}}
4039 \def\bbl@setdirs#1{% TODO - math
4040   \ifcase\bbl@select@type % TODO - strictly, not the right test
4041     \bbl@bodydir{#1}%
4042     \bbl@pardir{#1}%
4043   \fi
4044   \bbl@textdir{#1}}
4045 % TODO. Only if \bbl@bidimode > 0?:
4046 \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
4047 \DisableBabelHook{babel-bidi}
```

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

```
4048 \ifodd\bbl@engine   % luatex=1
4049 \else % pdftex=0, xetex=2
4050   \newcount\bbl@dirlevel
4051   \chardef\bbl@thetextdir\z@
4052   \chardef\bbl@thepardir\z@
4053   \def\bbl@textdir#1{%
4054     \ifcase#1\relax
4055       \chardef\bbl@thetextdir\z@
4056       \bbl@textdir@i\beginL\endL
4057     \else
4058       \chardef\bbl@thetextdir\@ne
4059       \bbl@textdir@i\beginR\endR
4060     \fi}
4061   \def\bbl@textdir@i#1#2{%
4062     \ifhmode
4063       \ifnum\currentgrouplevel>\z@
4064         \ifnum\currentgrouplevel=\bbl@dirlevel
4065           \bbl@error{Multiple bidi settings inside a group}%
4066             {I'll insert a new group, but expect wrong results.}%
4067           \bgroup\aftergroup#2\aftergroup\egroup
4068         \else
4069           \ifcase\currentgrouptype\or % 0 bottom
4070             \aftergroup#2% 1 simple {}
4071           \or
4072             \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
4073           \or
4074             \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
4075           \or\or\or % vbox vtop align
4076           \or
4077             \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
4078           \or\or\or\or\or\or % output math disc insert vcent mathchoice
4079           \or
4080             \aftergroup#2% 14 \begingroup
4081           \else
4082             \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
4083           \fi
4084         \fi
4085         \bbl@dirlevel\currentgrouplevel
4086       \fi
4087       #1%
4088     \fi}
4089   \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
4090   \let\bbl@bodydir\@gobble
4091   \let\bbl@pagedir\@gobble
4092   \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).

```
4093  \def\bbl@xebidipar{%
4094    \let\bbl@xebidipar\relax
4095    \TeXXeTstate\@ne
4096    \def\bbl@xeeverypar{%
4097      \ifcase\bbl@thepardir
4098        \ifcase\bbl@thetextdir\else\beginR\fi
4099      \else
4100        {\setbox\z@\lastbox\beginR\box\z@}%
4101      \fi}%
4102    \let\bbl@severypar\everypar
4103    \newtoks\everypar
4104    \everypar=\bbl@severypar
4105    \bbl@severypar{\bbl@xeeverypar\the\everypar}}
4106  \ifnum\bbl@bidimode>200
4107    \let\bbl@textdir@i\@gobbletwo
4108    \let\bbl@xebidipar\@empty
4109    \AddBabelHook{bidi}{foreign}{%
4110      \def\bbl@tempa{\def\BabelText####1}%
4111      \ifcase\bbl@thetextdir
4112        \expandafter\bbl@tempa\expandafter{\BabelText{\LR{##1}}}%
4113      \else
4114        \expandafter\bbl@tempa\expandafter{\BabelText{\RL{##1}}}%
4115      \fi}
4116    \def\bbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}
4117  \fi
4118 \fi
```

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

```
4119 \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}
4120 \AtBeginDocument{%
4121   \ifx\pdfstringdefDisableCommands\@undefined\else
4122     \ifx\pdfstringdefDisableCommands\relax\else
4123       \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
4124     \fi
4125   \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.

```
4126 \bbl@trace{Local Language Configuration}
4127 \ifx\loadlocalcfg\@undefined
4128   \@ifpackagewith{babel}{noconfigs}%
4129     {\let\loadlocalcfg\@gobble}%
4130     {\def\loadlocalcfg#1{%
4131       \InputIfFileExists{#1.cfg}%
4132         {\typeout{************************************^^J%
4133                   * Local config file #1.cfg used^^J%
4134                   *}}%
4135         \@empty}}
4136 \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).

```
4137 \bbl@trace{Language options}
4138 \let\bbl@afterlang\relax
4139 \let\BabelModifiers\relax
4140 \let\bbl@loaded\@empty
4141 \def\bbl@load@language#1{%
4142   \InputIfFileExists{#1.ldf}%
4143     {\edef\bbl@loaded{\CurrentOption
4144       \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}%
4145     \expandafter\let\expandafter\bbl@afterlang
4146       \csname\CurrentOption.ldf-h@@k\endcsname
4147     \expandafter\let\expandafter\BabelModifiers
4148       \csname bbl@mod@\CurrentOption\endcsname}%
4149     {\bbl@error{%
4150       Unknown option '\CurrentOption'. Either you misspelled it\\%
4151       or the language definition file \CurrentOption.ldf was not found}{%
4152       Valid options are, among others: shorthands=, KeepShorthandsActive,\\%
4153       activeacute, activegrave, noconfigs, safe=, main=, math=\\%
4154       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.

```
4155 \def\bbl@try@load@lang#1#2#3{%
4156   \IfFileExists{\CurrentOption.ldf}%
4157     {\bbl@load@language{\CurrentOption}}%
4158     {#1\bbl@load@language{#2}#3}}
4159 %
4160 \DeclareOption{hebrew}{%
4161   \input{rlbabel.def}%
4162   \bbl@load@language{hebrew}}
4163 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}{}}
4164 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}{}}
4165 \DeclareOption{nynorsk}{\bbl@try@load@lang{}{norsk}{}}
4166 \DeclareOption{polutonikogreek}{%
4167   \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}
4168 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}{}}
4169 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}{}}
4170 \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.

```
4171 \ifx\bbl@opt@config\@nnil
4172   \@ifpackagewith{babel}{noconfigs}{}%
4173     {\InputIfFileExists{bblopts.cfg}%
4174       {\typeout{***********************************^^J%
4175               * Local config file bblopts.cfg used^^J%
4176               *}}%
4177     {}}%
4178 \else
4179   \InputIfFileExists{\bbl@opt@config.cfg}%
4180     {\typeout{***********************************^^J%
4181              * Local config file \bbl@opt@config.cfg used^^J%
4182              *}}%
4183     {\bbl@error{%
4184       Local config file '\bbl@opt@config.cfg' not found}{%
4185       Perhaps you misspelled it.}}%
4186 \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.

```
4187 \ifx\bbl@opt@main\@nnil
4188   \ifnum\bbl@iniflag>\z@  % if all ldf's: set implicitly, no main pass
4189     \let\bbl@tempb\@empty
4190     \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}%
4191     \bbl@foreach\bbl@tempa{\edef\bbl@tempb{#1,\bbl@tempb}}%
4192     \bbl@foreach\bbl@tempb{%    \bbl@tempb is a reversed list
4193       \ifx\bbl@opt@main\@nnil % ie, if not yet assigned
4194         \ifodd\bbl@iniflag % = *=
4195           \IfFileExists{babel-#1.tex}{\def\bbl@opt@main{#1}}{}%
4196         \else % n +=
4197           \IfFileExists{#1.ldf}{\def\bbl@opt@main{#1}}{}%
4198         \fi
4199       \fi}%
4200   \fi
4201 \else
4202   \bbl@info{Main language set with 'main='. Except if you have\\%
4203           problems, prefer the default mechanism for setting\\%
4204           the main language. Reported}
4205 \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).

```
4206 \ifx\bbl@opt@main\@nnil\else
4207   \bbl@csarg\let{loadmain\expandafter}\csname ds@\bbl@opt@main\endcsname
4208   \expandafter\let\csname ds@\bbl@opt@main\endcsname\relax
4209 \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.

```
4210 \bbl@foreach\bbl@language@opts{%
4211   \def\bbl@tempa{#1}%
4212   \ifx\bbl@tempa\bbl@opt@main\else
4213     \ifnum\bbl@iniflag<\tw@    % 0 ø (other = ldf)
4214       \bbl@ifunset{ds@#1}%
4215         {\DeclareOption{#1}{\bbl@load@language{#1}}}%
4216         {}%
4217     \else                      % + * (other = ini)
4218       \DeclareOption{#1}{%
4219         \bbl@ldfinit
4220         \babelprovide[import]{#1}%
4221         \bbl@afterldf{}}%
4222     \fi
4223   \fi}
4224 \bbl@foreach\@classoptionslist{%
4225   \def\bbl@tempa{#1}%
4226   \ifx\bbl@tempa\bbl@opt@main\else
4227     \ifnum\bbl@iniflag<\tw@    % 0 ø (other = ldf)
4228       \bbl@ifunset{ds@#1}%
4229         {\IfFileExists{#1.ldf}%
4230           {\DeclareOption{#1}{\bbl@load@language{#1}}}%
4231           {}}%
4232         {}%
4233     \else                      % + * (other = ini)
4234       \IfFileExists{babel-#1.tex}%
4235         {\DeclareOption{#1}{%
4236           \bbl@ldfinit
4237           \babelprovide[import]{#1}%
4238           \bbl@afterldf{}}}%
4239         {}%
4240     \fi
```

148

```
4241  \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):

```
4242 \def\AfterBabelLanguage#1{%
4243   \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang}{}}
4244 \DeclareOption*{}
4245 \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.

```
4246 \bbl@trace{Option 'main'}
4247 \ifx\bbl@opt@main\@nnil
4248   \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}
4249   \let\bbl@tempc\@empty
4250   \bbl@for\bbl@tempb\bbl@tempa{%
4251     \bbl@xin@{,\bbl@tempb,}{,\bbl@loaded,}%
4252     \ifin@\edef\bbl@tempc{\bbl@tempb}\fi}
4253   \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
4254   \expandafter\bbl@tempa\bbl@loaded,\@nnil
4255   \ifx\bbl@tempb\bbl@tempc\else
4256     \bbl@warning{%
4257       Last declared language option is '\bbl@tempc',\\%
4258       but the last processed one was '\bbl@tempb'.\\%
4259       The main language can't be set as both a global\\%
4260       and a package option. Use 'main=\bbl@tempc' as\\%
4261       option. Reported}
4262   \fi
4263 \else
4264   \ifodd\bbl@iniflag  % case 1,3 (main is ini)
4265     \bbl@ldfinit
4266     \let\CurrentOption\bbl@opt@main
4267     \bbl@exp{%  \bbl@opt@provide = empty if *
4268       \\\babelprovide[\bbl@opt@provide,import,main]{\bbl@opt@main}}%
4269     \bbl@afterldf{}
4270     \DeclareOption{\bbl@opt@main}{}
4271   \else % case 0,2 (main is ldf)
4272     \ifx\bbl@loadmain\relax
4273       \DeclareOption{\bbl@opt@main}{\bbl@load@language{\bbl@opt@main}}
4274     \else
4275       \DeclareOption{\bbl@opt@main}{\bbl@loadmain}
4276     \fi
4277     \ExecuteOptions{\bbl@opt@main}
4278     \@namedef{ds@\bbl@opt@main}{}%
4279   \fi
4280   \DeclareOption*{}
4281   \ProcessOptions*
4282 \fi
4283 \def\AfterBabelLanguage{%
4284   \bbl@error
4285     {Too late for \string\AfterBabelLanguage}%
4286     {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.

```
4287 \ifx\bbl@main@language\@undefined
4288   \bbl@info{%
4289     You haven't specified a language. I'll use 'nil'\\%
```

```
4290    as the main language. Reported}
4291    \bbl@load@language{nil}
4292 \fi
4293 ⟨/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

```
4294 ⟨∗kernel⟩
4295 \let\bbl@onlyswitch\@empty
4296 \input babel.def
4297 \let\bbl@onlyswitch\@undefined
4298 ⟨/kernel⟩
4299 ⟨∗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.

```
4300 ⟨⟨Make sure ProvidesFile is defined⟩⟩
4301 \ProvidesFile{hyphen.cfg}[⟨⟨date⟩⟩ ⟨⟨version⟩⟩ Babel hyphens]
4302 \xdef\bbl@format{\jobname}
4303 \def\bbl@version{⟨⟨version⟩⟩}
4304 \def\bbl@date{⟨⟨date⟩⟩}
4305 \ifx\AtBeginDocument\@undefined
4306    \def\@empty{}
4307 \fi
4308 ⟨⟨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.

```
4309 \def\process@line#1#2 #3 #4 {%
4310    \ifx=#1%
4311       \process@synonym{#2}%
4312    \else
4313       \process@language{#1#2}{#3}{#4}%
4314    \fi
4315    \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.

```
4316 \toks@{}
4317 \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.

```
4318 \def\process@synonym#1{%
4319   \ifnum\last@language=\m@ne
4320     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4321   \else
4322     \expandafter\chardef\csname l@#1\endcsname\last@language
4323     \wlog{\string\l@#1=\string\language\the\last@language}%
4324     \expandafter\let\csname #1hyphenmins\expandafter\endcsname
4325       \csname\languagename hyphenmins\endcsname
4326     \let\bbl@elt\relax
4327     \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}{}{}}%
4328   \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.

```
4329 \def\process@language#1#2#3{%
4330   \expandafter\addlanguage\csname l@#1\endcsname
4331   \expandafter\language\csname l@#1\endcsname
4332   \edef\languagename{#1}%
4333   \bbl@hook@everylanguage{#1}%
4334   %  > luatex
4335   \bbl@get@enc#1::\@@@
4336   \begingroup
4337     \lefthyphenmin\m@ne
4338     \bbl@hook@loadpatterns{#2}%
4339     %  > luatex
4340     \ifnum\lefthyphenmin=\m@ne
4341     \else
4342       \expandafter\xdef\csname #1hyphenmins\endcsname{%
4343         \the\lefthyphenmin\the\righthyphenmin}%
4344     \fi
4345   \endgroup
4346   \def\bbl@tempa{#3}%
4347   \ifx\bbl@tempa\@empty\else
4348     \bbl@hook@loadexceptions{#3}%
4349     %  > luatex
4350   \fi
4351   \let\bbl@elt\relax
```

```
4352    \edef\bbl@languages{%
4353      \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
4354    \ifnum\the\language=\z@
4355      \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4356        \set@hyphenmins\tw@\thr@@\relax
4357      \else
4358        \expandafter\expandafter\expandafter\set@hyphenmins
4359          \csname #1hyphenmins\endcsname
4360      \fi
4361      \the\toks@
4362      \toks@{}%
4363    \fi}
```

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

```
4364 \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.

```
4365 \def\bbl@hook@everylanguage#1{}
4366 \def\bbl@hook@loadpatterns#1{\input #1\relax}
4367 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
4368 \def\bbl@hook@loadkernel#1{%
4369    \def\addlanguage{\csname newlanguage\endcsname}%
4370    \def\adddialect##1##2{%
4371      \global\chardef##1##2\relax
4372      \wlog{\string##1 = a dialect from \string\language##2}}%
4373    \def\iflanguage##1{%
4374      \expandafter\ifx\csname l@##1\endcsname\relax
4375        \@nolanerr{##1}%
4376      \else
4377        \ifnum\csname l@##1\endcsname=\language
4378          \expandafter\expandafter\expandafter\@firstoftwo
4379        \else
4380          \expandafter\expandafter\expandafter\@secondoftwo
4381        \fi
4382      \fi}%
4383    \def\providehyphenmins##1##2{%
4384      \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
4385        \@namedef{##1hyphenmins}{##2}%
4386      \fi}%
4387    \def\set@hyphenmins##1##2{%
4388      \lefthyphenmin##1\relax
4389      \righthyphenmin##2\relax}%
4390    \def\selectlanguage{%
4391      \errhelp{Selecting a language requires a package supporting it}%
4392      \errmessage{Not loaded}}%
4393    \let\foreignlanguage\selectlanguage
4394    \let\otherlanguage\selectlanguage
4395    \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4396    \def\bbl@usehooks##1##2{}% TODO. Temporary!!
4397    \def\setlocale{%
4398      \errhelp{Find an armchair, sit down and wait}%
4399      \errmessage{Not yet available}}%
4400    \let\uselocale\setlocale
4401    \let\locale\setlocale
4402    \let\selectlocale\setlocale
4403    \let\localename\setlocale
4404    \let\textlocale\setlocale
4405    \let\textlanguage\setlocale
4406    \let\languagetext\setlocale}
4407 \begingroup
```

152

```
4408  \def\AddBabelHook#1#2{%
4409    \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
4410      \def\next{\toks1}%
4411    \else
4412      \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname####1}%
4413    \fi
4414    \next}
4415  \ifx\directlua\@undefined
4416    \ifx\XeTeXinputencoding\@undefined\else
4417      \input xebabel.def
4418    \fi
4419  \else
4420    \input luababel.def
4421  \fi
4422  \openin1 = babel-\bbl@format.cfg
4423  \ifeof1
4424  \else
4425    \input babel-\bbl@format.cfg\relax
4426  \fi
4427  \closein1
4428  \endgroup
4429  \bbl@hook@loadkernel{switch.def}
```

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

```
4430 \openin1 = language.dat
```

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

```
4431 \def\languagename{english}%
4432 \ifeof1
4433   \message{I couldn't find the file language.dat,\space
4434           I will try the file hyphen.tex}
4435   \input hyphen.tex\relax
4436   \chardef\l@english\z@
4437 \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.

```
4438   \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.

```
4439   \loop
4440     \endlinechar\m@ne
4441     \read1 to \bbl@line
4442     \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.

```
4443     \if T\ifeof1F\fi T\relax
4444       \ifx\bbl@line\@empty\else
4445         \edef\bbl@line{\bbl@line\space\space\space}%
4446         \expandafter\process@line\bbl@line\relax
4447       \fi
4448   \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.

```
4449   \begingroup
4450     \def\bbl@elt#1#2#3#4{%
```

```
4451      \global\language=#2\relax
4452      \gdef\languagename{#1}%
4453      \def\bbl@elt##1##2##3##4{}}%
4454    \bbl@languages
4455  \endgroup
4456 \fi
4457 \closein1
```

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

```
4458 \if/\the\toks@/\else
4459   \errhelp{language.dat loads no language, only synonyms}
4460   \errmessage{Orphan language synonym}
4461 \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.

```
4462 \let\bbl@line\@undefined
4463 \let\process@line\@undefined
4464 \let\process@synonym\@undefined
4465 \let\process@language\@undefined
4466 \let\bbl@get@enc\@undefined
4467 \let\bbl@hyph@enc\@undefined
4468 \let\bbl@tempa\@undefined
4469 \let\bbl@hook@loadkernel\@undefined
4470 \let\bbl@hook@everylanguage\@undefined
4471 \let\bbl@hook@loadpatterns\@undefined
4472 \let\bbl@hook@loadexceptions\@undefined
4473 ⟨/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].

```
4474 ⟨⟨*More package options⟩⟩ ≡
4475 \chardef\bbl@bidimode\z@
4476 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4477 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
4478 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4479 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4480 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4481 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4482 ⟨⟨/More package options⟩⟩
```

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

At the time of this writing, fontspec shows a warning about there are languages not available, which some people think refers to babel, even if there is nothing wrong. Here is hack to patch fontspec to avoid the misleading message, which is replaced ba a more explanatory one.

```
4483 ⟨⟨*Font selection⟩⟩ ≡
4484 \bbl@trace{Font handling with fontspec}
4485 \ifx\ExplSyntaxOn\@undefined\else
4486   \def\bbl@fs@warn#1#2{% \bbl@tempfs is the original macro
4487     \in@{,#1,}{,no-script,language-not-exist,}%
4488     \ifin@\else\bbl@tempfs{#1}{#2}\fi}
4489 \def\bbl@loadfontspec{%
4490   \let\bbl@loadfontspec\relax
4491   \ifx\fontspec\@undefined
4492     \usepackage{fontspec}%
4493   \fi}%
```

```
4494 \fi
4495 \@onlypreamble\babelfont
4496 \newcommand\babelfont[2][]{%  1=langs/scripts 2=fam
4497   \bbl@foreach{#1}{%
4498     \expandafter\ifx\csname date##1\endcsname\relax
4499       \IfFileExists{babel-##1.tex}%
4500         {\babelprovide{##1}}%
4501         {}%
4502     \fi}%
4503   \edef\bbl@tempa{#1}%
4504   \def\bbl@tempb{#2}%  Used by \bbl@bblfont
4505   \bbl@loadfontspec
4506   \EnableBabelHook{babel-fontspec}% Just calls \bbl@switchfont
4507   \bbl@bblfont}
4508 \newcommand\bbl@bblfont[2][]{% 1=features 2=fontname, @font=rm|sf|tt
4509   \bbl@ifunset{\bbl@tempb family}%
4510     {\bbl@providefam{\bbl@tempb}}%
4511     {}%
4512 % For the default font, just in case:
4513   \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
4514   \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
4515     {\bbl@csarg\edef{\bbl@tempb dflt@}{<>{#1}{#2}}% save bbl@rmdflt@
4516     \bbl@exp{%
4517       \let\<bbl@\bbl@tempb dflt@\languagename>\<bbl@\bbl@tempb dflt@>%
4518       \\\bbl@font@set\<bbl@\bbl@tempb dflt@\languagename>%
4519                      \<\bbl@tempb default>\<\bbl@tempb family>}}%
4520     {\bbl@foreach\bbl@tempa{% ie bbl@rmdflt@lang / *scrt
4521       \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:

```
4522 \def\bbl@providefam#1{%
4523   \bbl@exp{%
4524     \\\newcommand\<#1default>{}% Just define it
4525     \\\bbl@add@list\\\bbl@font@fams{#1}%
4526     \\\DeclareRobustCommand\<#1family>{%
4527       \\\not@math@alphabet\<#1family>\relax
4528     % \\\prepare@family@series@update{#1}\<#1default>% TODO. Fails
4529       \\\fontfamily\<#1default>%
4530       \<\ifx>\\\UseHooks\\\@undefined\<\else>\\\UseHook{#1family}\<\fi>%
4531       \\\selectfont}%
4532     \\\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.

```
4533 \def\bbl@nostdfont#1{%
4534   \bbl@ifunset{bbl@WFF@\f@family}%
4535     {\bbl@csarg\gdef{WFF@\f@family}{}%  Flag, to avoid dupl warns
4536      \bbl@infowarn{The current font is not a babel standard family:\\%
4537        #1%
4538        \fontname\font\\%
4539        There is nothing intrinsically wrong with this warning, and\\%
4540        you can ignore it altogether if you do not need these\\%
4541        families. But if they are used in the document, you should be\\%
4542        aware 'babel' will not set Script and Language for them, so\\%
4543        you may consider defining a new family with \string\babelfont.\\%
4544        See the manual for further details about \string\babelfont.\\%
4545        Reported}}
4546     {}}%
4547 \gdef\bbl@switchfont{%
4548   \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
4549   \bbl@exp{%  eg Arabic -> arabic
4550     \lowercase{\edef\\\bbl@tempa{\bbl@cl{sname}}}}%
4551   \bbl@foreach\bbl@font@fams{%
4552     \bbl@ifunset{bbl@##1dflt@\languagename}%     (1) language?
```

```
4553       {\bbl@ifunset{bbl@##1dflt@*\bbl@tempa}%     (2) from script?
4554         {\bbl@ifunset{bbl@##1dflt@}%               2=F - (3) from generic?
4555            {}%                                     123=F - nothing!
4556            {\bbl@exp{%                             3=T - from generic
4557               \global\let\<bbl@##1dflt@\languagename>%
4558                         \<bbl@##1dflt@>}}}%
4559         {\bbl@exp{%                                2=T - from script
4560            \global\let\<bbl@##1dflt@\languagename>%
4561                      \<bbl@##1dflt@*\bbl@tempa>}}}%
4562       {}}%                                         1=T - language, already defined
4563 \def\bbl@tempa{\bbl@nostdfont{}}%
4564 \bbl@foreach\bbl@font@fams{%      don't gather with prev for
4565   \bbl@ifunset{bbl@##1dflt@\languagename}%
4566     {\bbl@cs{famrst@##1}%
4567      \global\bbl@csarg\let{famrst@##1}\relax}%
4568     {\bbl@exp{% order is relevant. TODO: but sometimes wrong!
4569        \\\bbl@add\\\originalTeX{%
4570          \\\bbl@font@rst{\bbl@cl{##1dflt}}%
4571                      \<##1default>\<##1family>{##1}}%
4572        \\\bbl@font@set\<bbl@##1dflt@\languagename>% the main part!
4573                      \<##1default>\<##1family>}}}%
4574 \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.

```
4575 \ifx\f@family\@undefined\else   % if latex
4576   \ifcase\bbl@engine          % if pdftex
4577     \let\bbl@ckeckstdfonts\relax
4578   \else
4579     \def\bbl@ckeckstdfonts{%
4580       \begingroup
4581         \global\let\bbl@ckeckstdfonts\relax
4582         \let\bbl@tempa\@empty
4583         \bbl@foreach\bbl@font@fams{%
4584           \bbl@ifunset{bbl@##1dflt@}%
4585             {\@nameuse{##1family}%
4586              \bbl@csarg\gdef{WFF@\f@family}{}% Flag
4587              \bbl@exp{\\\bbl@add\\\bbl@tempa{* \<##1family>= \f@family\\\\%
4588                \space\space\fontname\font\\\\}}%
4589              \bbl@csarg\xdef{##1dflt@}{\f@family}%
4590              \expandafter\xdef\csname ##1default\endcsname{\f@family}}%
4591             {}}%
4592         \ifx\bbl@tempa\@empty\else
4593           \bbl@infowarn{The following font families will use the default\\%
4594             settings for all or some languages:\\%
4595             \bbl@tempa
4596             There is nothing intrinsically wrong with it, but\\%
4597             'babel' will no set Script and Language, which could\\%
4598             be relevant in some languages. If your document uses\\%
4599             these families, consider redefining them with \string\babelfont.\\%
4600             Reported}%
4601         \fi
4602       \endgroup}
4603   \fi
4604 \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.

```
4605 \def\bbl@font@set#1#2#3{% eg \bbl@rmdflt@lang \rmdefault \rmfamily
4606   \bbl@xin@{<>}{#1}%
4607   \ifin@
4608     \bbl@exp{\\\bbl@fontspec@set\\#1\expandafter\@gobbletwo#1\\#3}%
```

156

```
4609  \fi
4610  \bbl@exp{%                'Unprotected' macros return prev values
4611    \def\\#2{#1}%           eg, \rmdefault{\bbl@rmdflt@lang}
4612    \\\bbl@ifsamestring{#2}{\f@family}%
4613      {\\#3%
4614        \\\bbl@ifsamestring{\f@series}{\bfdefault}{\\\bfseries}{}%
4615       \let\\\bbl@tempa\relax}%
4616      {}}}
4617 %     TODO - next should be global?, but even local does its job. I'm
4618 %     still not sure -- must investigate:
4619 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
4620    \let\bbl@tempe\bbl@mapselect
4621    \let\bbl@mapselect\relax
4622    \let\bbl@temp@fam#4%          eg, '\rmfamily', to be restored below
4623    \let#4\@empty       %        Make sure \renewfontfamily is valid
4624    \bbl@exp{%
4625      \let\\\bbl@temp@pfam\<\bbl@stripslash#4\space>% eg, '\rmfamily '
4626      \<keys_if_exist:nnF>{fontspec-opentype}{Script/\bbl@cl{sname}}%
4627        {\\\newfontscript{\bbl@cl{sname}}{\bbl@cl{sotf}}}%
4628      \<keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@cl{lname}}%
4629        {\\\newfontlanguage{\bbl@cl{lname}}{\bbl@cl{lotf}}}%
4630      \let\\\bbl@tempfs\<__fontspec_warning:nx>%
4631      \let\<__fontspec_warning:nx>\\\bbl@fs@warn
4632      \\\renewfontfamily\\#4%
4633        [\bbl@cl{lsys},#2]}{#3}% ie \bbl@exp{..}{#3}
4634    \bbl@exp{\let\<__fontspec_warning:nx>\\\bbl@tempfs}%
4635    \begingroup
4636      #4%
4637      \xdef#1{\f@family}%     eg, \bbl@rmdflt@lang{FreeSerif(0)}
4638    \endgroup
4639    \let#4\bbl@temp@fam
4640    \bbl@exp{\let\<\bbl@stripslash#4\space>}\bbl@temp@pfam
4641    \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.

```
4642 \def\bbl@font@rst#1#2#3#4{%
4643    \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.

```
4644 \def\bbl@font@fams{rm,sf,tt}
4645 ⟨⟨/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.

```
4646 ⟨⟨*Footnote changes⟩⟩ ≡
4647 \bbl@trace{Bidi footnotes}
4648 \ifnum\bbl@bidimode>\z@
4649   \def\bbl@footnote#1#2#3{%
4650     \@ifnextchar[%
4651       {\bbl@footnote@o{#1}{#2}{#3}}%
4652       {\bbl@footnote@x{#1}{#2}{#3}}}
4653   \long\def\bbl@footnote@x#1#2#3#4{%
4654     \bgroup
4655       \select@language@x{\bbl@main@language}%
4656       \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
4657     \egroup}
4658   \long\def\bbl@footnote@o#1#2#3[#4]#5{%
```

```
4659      \bgroup
4660        \select@language@x{\bbl@main@language}%
4661        \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
4662      \egroup}
4663    \def\bbl@footnotetext#1#2#3{%
4664      \@ifnextchar[%
4665        {\bbl@footnotetext@o{#1}{#2}{#3}}%
4666        {\bbl@footnotetext@x{#1}{#2}{#3}}}
4667    \long\def\bbl@footnotetext@x#1#2#3#4{%
4668      \bgroup
4669        \select@language@x{\bbl@main@language}%
4670        \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
4671      \egroup}
4672    \long\def\bbl@footnotetext@o#1#2#3[#4]#5{%
4673      \bgroup
4674        \select@language@x{\bbl@main@language}%
4675        \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
4676      \egroup}
4677    \def\BabelFootnote#1#2#3#4{%
4678      \ifx\bbl@fn@footnote\@undefined
4679        \let\bbl@fn@footnote\footnote
4680      \fi
4681      \ifx\bbl@fn@footnotetext\@undefined
4682        \let\bbl@fn@footnotetext\footnotetext
4683      \fi
4684      \bbl@ifblank{#2}%
4685        {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}%
4686         \@namedef{\bbl@stripslash#1text}%
4687           {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
4688        {\def#1{\bbl@exp{\\\bbl@footnote{\\\foreignlanguage{#2}}}{#3}{#4}}%
4689         \@namedef{\bbl@stripslash#1text}%
4690           {\bbl@exp{\\\bbl@footnotetext{\\\foreignlanguage{#2}}}{#3}{#4}}}}
4691  \fi
4692  ⟨⟨/Footnote changes⟩⟩
```

Now, the code.

```
4693  ⟨∗xetex⟩
4694  \def\BabelStringsDefault{unicode}
4695  \let\xebbl@stop\relax
4696  \AddBabelHook{xetex}{encodedcommands}{%
4697    \def\bbl@tempa{#1}%
4698    \ifx\bbl@tempa\@empty
4699      \XeTeXinputencoding"bytes"%
4700    \else
4701      \XeTeXinputencoding"#1"%
4702    \fi
4703    \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4704  \AddBabelHook{xetex}{stopcommands}{%
4705    \xebbl@stop
4706    \let\xebbl@stop\relax}
4707  \def\bbl@intraspace#1 #2 #3\@@{%
4708    \bbl@csarg\gdef{xeisp@\languagename}%
4709      {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4710  \def\bbl@intrapenalty#1\@@{%
4711    \bbl@csarg\gdef{xeipn@\languagename}%
4712      {\XeTeXlinebreakpenalty #1\relax}}
4713  \def\bbl@provide@intraspace{%
4714    \bbl@xin@{/s}{/\bbl@cl{lnbrk}}%
4715    \ifin@\else\bbl@xin@{/c}{/\bbl@cl{lnbrk}}\fi
4716    \ifin@
4717      \bbl@ifunset{bbl@intsp@\languagename}{}%
4718        {\expandafter\ifx\csname bbl@intsp@\languagename\endcsname\@empty\else
4719          \ifx\bbl@KVP@intraspace\@nnil
```

```
4720            \bbl@exp{%
4721                \\\bbl@intraspace\bbl@cl{intsp}\\\@@}%
4722          \fi
4723          \ifx\bbl@KVP@intrapenalty\@nnil
4724            \bbl@intrapenalty0\@@
4725          \fi
4726        \fi
4727        \ifx\bbl@KVP@intraspace\@nnil\else % We may override the ini
4728          \expandafter\bbl@intraspace\bbl@KVP@intraspace\@@
4729        \fi
4730        \ifx\bbl@KVP@intrapenalty\@nnil\else
4731          \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
4732        \fi
4733        \bbl@exp{%
4734          % TODO. Execute only once (but redundant):
4735          \\\bbl@add\<extras\languagename>{%
4736            \XeTeXlinebreaklocale "\bbl@cl{tbcp}"%
4737            \<bbl@xeisp@\languagename>%
4738            \<bbl@xeipn@\languagename>}%
4739          \\\bbl@toglobal\<extras\languagename>%
4740          \\\bbl@add\<noextras\languagename>{%
4741            \XeTeXlinebreaklocale "en"}%
4742          \\\bbl@toglobal\<noextras\languagename>}%
4743        \ifx\bbl@ispacesize\@undefined
4744          \gdef\bbl@ispacesize{\bbl@cl{xeisp}}%
4745          \ifx\AtBeginDocument\@notprerr
4746            \expandafter\@secondoftwo  % to execute right now
4747          \fi
4748          \AtBeginDocument{\bbl@patchfont{\bbl@ispacesize}}%
4749        \fi}%
4750    \fi}
4751 \ifx\DisableBabelHook\@undefined\endinput\fi
4752 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4753 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
4754 \DisableBabelHook{babel-fontspec}
4755 ⟨⟨Font selection⟩⟩
4756 \input txtbabel.def
4757 ⟨/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.

```
4758 ⟨∗texxet⟩
4759 \providecommand\bbl@provide@intraspace{}
4760 \bbl@trace{Redefinitions for bidi layout}
4761 \def\bbl@sspre@caption{%
4762   \bbl@exp{\everyhbox{\\\bbl@textdir\bbl@cs{wdir@\bbl@main@language}}}}
4763 \ifx\bbl@opt@layout\@nnil\endinput\fi  % No layout
4764 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
4765 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
4766 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
4767   \def\@hangfrom#1{%
4768     \setbox\@tempboxa\hbox{{#1}}%
4769     \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
4770     \noindent\box\@tempboxa}
4771   \def\raggedright{%
4772     \let\\\@centercr
```

```
4773      \bbl@startskip\z@skip
4774      \@rightskip\@flushglue
4775      \bbl@endskip\@rightskip
4776      \parindent\z@
4777      \parfillskip\bbl@startskip}
4778   \def\raggedleft{%
4779      \let\\\@centercr
4780      \bbl@startskip\@flushglue
4781      \bbl@endskip\z@skip
4782      \parindent\z@
4783      \parfillskip\bbl@endskip}
4784 \fi
4785 \IfBabelLayout{lists}
4786   {\bbl@sreplace\list
4787      {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
4788    \def\bbl@listleftmargin{%
4789      \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
4790    \ifcase\bbl@engine
4791      \def\labelenumii){\theenumii(}% pdftex doesn't reverse ()
4792      \def\p@enumiii{\p@enumii)\theenumii(}%
4793    \fi
4794    \bbl@sreplace\@verbatim
4795      {\leftskip\@totalleftmargin}%
4796      {\bbl@startskip\textwidth
4797       \advance\bbl@startskip-\linewidth}%
4798    \bbl@sreplace\@verbatim
4799      {\rightskip\z@skip}%
4800      {\bbl@endskip\z@skip}}%
4801   {}
4802 \IfBabelLayout{contents}
4803   {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
4804    \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
4805   {}
4806 \IfBabelLayout{columns}
4807   {\bbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbl@outputhbox}%
4808    \def\bbl@outputhbox#1{%
4809      \hb@xt@\textwidth{%
4810        \hskip\columnwidth
4811        \hfil
4812        {\normalcolor\vrule \@width\columnseprule}%
4813        \hfil
4814        \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
4815        \hskip-\textwidth
4816        \hb@xt@\columnwidth{\box\@outputbox \hss}%
4817        \hskip\columnsep
4818        \hskip\columnwidth}}}%
4819   {}
4820 ⟨⟨Footnote changes⟩⟩
4821 \IfBabelLayout{footnotes}%
4822   {\BabelFootnote\footnote\languagename{}{}%
4823    \BabelFootnote\localfootnote\languagename{}{}%
4824    \BabelFootnote\mainfootnote{}{}{}}
4825   {}
```

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.

```
4826 \IfBabelLayout{counters}%
4827   {\let\bbl@latinarabic=\@arabic
4828    \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
4829    \let\bbl@asciiroman=\@roman
4830    \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciiroman#1}}}%
4831    \let\bbl@asciiRoman=\@Roman
4832    \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}}{}
```

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`).

```
4834 ⟨*luatex⟩
4835 \ifx\AddBabelHook\@undefined % When plain.def, babel.sty starts
4836 \bbl@trace{Read language.dat}
4837 \ifx\bbl@readstream\@undefined
4838   \csname newread\endcsname\bbl@readstream
4839 \fi
4840 \begingroup
4841   \toks@{}
4842   \count@\z@ % 0=start, 1=0th, 2=normal
4843   \def\bbl@process@line#1#2 #3 #4 {%
4844     \ifx=#1%
4845       \bbl@process@synonym{#2}%
4846     \else
4847       \bbl@process@language{#1#2}{#3}{#4}%
4848     \fi
4849     \ignorespaces}
4850   \def\bbl@manylang{%
4851     \ifnum\bbl@last>\@ne
4852       \bbl@info{Non-standard hyphenation setup}%
4853     \fi
4854     \let\bbl@manylang\relax}
4855   \def\bbl@process@language#1#2#3{%
4856     \ifcase\count@
4857       \@ifundefined{zth@#1}{\count@\tw@}{\count@\@ne}%
4858     \or
4859       \count@\tw@
4860     \fi
```

```
4861    \ifnum\count@=\tw@
4862      \expandafter\addlanguage\csname l@#1\endcsname
4863      \language\allocationnumber
4864      \chardef\bbl@last\allocationnumber
4865      \bbl@manylang
4866      \let\bbl@elt\relax
4867      \xdef\bbl@languages{%
4868        \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
4869    \fi
4870    \the\toks@
4871    \toks@{}}
4872 \def\bbl@process@synonym@aux#1#2{%
4873    \global\expandafter\chardef\csname l@#1\endcsname#2\relax
4874    \let\bbl@elt\relax
4875    \xdef\bbl@languages{%
4876      \bbl@languages\bbl@elt{#1}{#2}{}{}}}%
4877 \def\bbl@process@synonym#1{%
4878    \ifcase\count@
4879      \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
4880    \or
4881      \@ifundefined{zth@#1}{\bbl@process@synonym@aux{#1}{0}}{}%
4882    \else
4883      \bbl@process@synonym@aux{#1}{\the\bbl@last}%
4884    \fi}
4885 \ifx\bbl@languages\@undefined % Just a (sensible?) guess
4886    \chardef\l@english\z@
4887    \chardef\l@USenglish\z@
4888    \chardef\bbl@last\z@
4889    \global\@namedef{bbl@hyphendata@0}{{hyphen.tex}{}}
4890    \gdef\bbl@languages{%
4891      \bbl@elt{english}{0}{hyphen.tex}{}%
4892      \bbl@elt{USenglish}{0}{}{}}
4893 \else
4894    \global\let\bbl@languages@format\bbl@languages
4895    \def\bbl@elt#1#2#3#4{% Remove all except language 0
4896      \ifnum#2>\z@\else
4897        \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
4898      \fi}%
4899    \xdef\bbl@languages{\bbl@languages}%
4900 \fi
4901 \def\bbl@elt#1#2#3#4{\@namedef{zth@#1}{}} % Define flags
4902 \bbl@languages
4903 \openin\bbl@readstream=language.dat
4904 \ifeof\bbl@readstream
4905    \bbl@warning{I couldn't find language.dat. No additional\\%
4906                patterns loaded. Reported}%
4907 \else
4908    \loop
4909      \endlinechar\m@ne
4910      \read\bbl@readstream to \bbl@line
4911      \endlinechar`\^^M
4912      \if T\ifeof\bbl@readstream F\fi T\relax
4913        \ifx\bbl@line\@empty\else
4914          \edef\bbl@line{\bbl@line\space\space\space}%
4915          \expandafter\bbl@process@line\bbl@line\relax
4916        \fi
4917    \repeat
4918 \fi
4919 \endgroup
4920 \bbl@trace{Macros for reading patterns files}
4921 \def\bbl@get@enc#1:#2:#3\@@@{\def\bbl@hyph@enc{#2}}
4922 \ifx\babelcatcodetablenum\@undefined
4923    \ifx\newcatcodetable\@undefined
```

162

```
4924      \def\babelcatcodetablenum{5211}
4925      \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4926    \else
4927      \newcatcodetable\babelcatcodetablenum
4928      \newcatcodetable\bbl@pattcodes
4929    \fi
4930  \else
4931    \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4932  \fi
4933  \def\bbl@luapatterns#1#2{%
4934    \bbl@get@enc#1::\@@@
4935    \setbox\z@\hbox\bgroup
4936      \begingroup
4937        \savecatcodetable\babelcatcodetablenum\relax
4938        \initcatcodetable\bbl@pattcodes\relax
4939        \catcodetable\bbl@pattcodes\relax
4940          \catcode`\#=6  \catcode`\$=3 \catcode`\&=4 \catcode`\^=7
4941          \catcode`\_=8  \catcode`\{=1 \catcode`\}=2 \catcode`\~=13
4942          \catcode`\@=11 \catcode`\^^I=10 \catcode`\^^J=12
4943          \catcode`\<=12 \catcode`\>=12 \catcode`\*=12 \catcode`\.=12
4944          \catcode`\-=12 \catcode`\/=12 \catcode`\[=12 \catcode`\]=12
4945          \catcode`\`=12 \catcode`\'=12 \catcode`\"=12
4946          \input #1\relax
4947        \catcodetable\babelcatcodetablenum\relax
4948      \endgroup
4949      \def\bbl@tempa{#2}%
4950      \ifx\bbl@tempa\@empty\else
4951        \input #2\relax
4952      \fi
4953    \egroup}%
4954  \def\bbl@patterns@lua#1{%
4955    \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
4956      \csname l@#1\endcsname
4957      \edef\bbl@tempa{#1}%
4958    \else
4959      \csname l@#1:\f@encoding\endcsname
4960      \edef\bbl@tempa{#1:\f@encoding}%
4961    \fi\relax
4962    \@namedef{lu@texhyphen@loaded@\the\language}{}% Temp
4963    \@ifundefined{bbl@hyphendata@\the\language}%
4964      {\def\bbl@elt##1##2##3##4{%
4965        \ifnum##2=\csname l@\bbl@tempa\endcsname % #2=spanish, dutch:OT1...
4966          \def\bbl@tempb{##3}%
4967          \ifx\bbl@tempb\@empty\else % if not a synonymous
4968            \def\bbl@tempc{{##3}{##4}}%
4969          \fi
4970          \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
4971        \fi}%
4972       \bbl@languages
4973       \@ifundefined{bbl@hyphendata@\the\language}%
4974         {\bbl@info{No hyphenation patterns were set for\\%
4975                   language '\bbl@tempa'. Reported}}%
4976         {\expandafter\expandafter\expandafter\bbl@luapatterns
4977           \csname bbl@hyphendata@\the\language\endcsname}}{}}
4978  \endinput\fi
4979    % Here ends \ifx\AddBabelHook\@undefined
4980    % A few lines are only read by hyphen.cfg
4981  \ifx\DisableBabelHook\@undefined
4982  \AddBabelHook{luatex}{everylanguage}{%
4983    \def\process@language##1##2##3{%
4984      \def\process@line####1####2 ####3 ####4 {}}}
4985  \AddBabelHook{luatex}{loadpatterns}{%
4986      \input #1\relax
```

```
4987     \expandafter\gdef\csname bbl@hyphendata@\the\language\endcsname
4988       {{#1}{}}}
4989 \AddBabelHook{luatex}{loadexceptions}{%
4990     \input #1\relax
4991     \def\bbl@tempb##1##2{{##1}{#1}}%
4992     \expandafter\xdef\csname bbl@hyphendata@\the\language\endcsname
4993       {\expandafter\expandafter\expandafter\bbl@tempb
4994       \csname bbl@hyphendata@\the\language\endcsname}}
4995 \endinput\fi
4996  % Here stops reading code for hyphen.cfg
4997  % The following is read the 2nd time it's loaded
4998 \begingroup  % TODO - to a lua file
4999 \catcode`\%=12
5000 \catcode`\'=12
5001 \catcode`\"=12
5002 \catcode`\:=12
5003 \directlua{
5004  Babel = Babel or {}
5005  function Babel.bytes(line)
5006    return line:gsub("(.)",
5007      function (chr) return unicode.utf8.char(string.byte(chr)) end)
5008  end
5009  function Babel.begin_process_input()
5010    if luatexbase and luatexbase.add_to_callback then
5011      luatexbase.add_to_callback('process_input_buffer',
5012                                Babel.bytes,'Babel.bytes')
5013    else
5014      Babel.callback = callback.find('process_input_buffer')
5015      callback.register('process_input_buffer',Babel.bytes)
5016    end
5017  end
5018  function Babel.end_process_input ()
5019    if luatexbase and luatexbase.remove_from_callback then
5020      luatexbase.remove_from_callback('process_input_buffer','Babel.bytes')
5021    else
5022      callback.register('process_input_buffer',Babel.callback)
5023    end
5024  end
5025  function Babel.addpatterns(pp, lg)
5026    local lg = lang.new(lg)
5027    local pats = lang.patterns(lg) or ''
5028    lang.clear_patterns(lg)
5029    for p in pp:gmatch('[^%s]+') do
5030      ss = ''
5031      for i in string.utfcharacters(p:gsub('%d', '')) do
5032        ss = ss .. '%d?' .. i
5033      end
5034      ss = ss:gsub('^%%d%?%.', '%%.') .. '%d?'
5035      ss = ss:gsub('%.%%d%?$', '%%.')
5036      pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
5037      if n == 0 then
5038        tex.sprint(
5039          [[\string\csname\space bbl@info\endcsname{New pattern: ]]
5040          .. p .. [[}]])
5041        pats = pats .. ' ' .. p
5042      else
5043        tex.sprint(
5044          [[\string\csname\space bbl@info\endcsname{Renew pattern: ]]
5045          .. p .. [[}]])
5046      end
5047    end
5048    lang.patterns(lg, pats)
5049  end
```

```
5050    Babel.characters = Babel.characters or {}
5051    Babel.ranges = Babel.ranges or {}
5052    function Babel.hlist_has_bidi(head)
5053      local has_bidi = false
5054      local ranges = Babel.ranges
5055      for item in node.traverse(head) do
5056        if item.id == node.id'glyph' then
5057          local itemchar = item.char
5058          local chardata = Babel.characters[itemchar]
5059          local dir = chardata and chardata.d or nil
5060          if not dir then
5061            for nn, et in ipairs(ranges) do
5062              if itemchar < et[1] then
5063                break
5064              elseif itemchar <= et[2] then
5065                dir = et[3]
5066                break
5067              end
5068            end
5069          end
5070          if dir and (dir == 'al' or dir == 'r') then
5071            has_bidi = true
5072          end
5073        end
5074      end
5075      return has_bidi
5076    end
5077    function Babel.set_chranges_b (script, chrng)
5078      if chrng == '' then return end
5079      texio.write('Replacing ' .. script .. ' script ranges')
5080      Babel.script_blocks[script] = {}
5081      for s, e in string.gmatch(chrng..' ', '(.-)%.%.(.-)%s') do
5082        table.insert(
5083          Babel.script_blocks[script], {tonumber(s,16), tonumber(e,16)})
5084      end
5085    end
5086 }
5087 \endgroup
5088 \ifx\newattribute\@undefined\else
5089   \newattribute\bbl@attr@locale
5090   \directlua{ Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale' }
5091   \AddBabelHook{luatex}{beforeextras}{%
5092     \setattribute\bbl@attr@locale\localeid}
5093 \fi
5094 \def\BabelStringsDefault{unicode}
5095 \let\luabbl@stop\relax
5096 \AddBabelHook{luatex}{encodedcommands}{%
5097   \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
5098   \ifx\bbl@tempa\bbl@tempb\else
5099     \directlua{Babel.begin_process_input()}%
5100     \def\luabbl@stop{%
5101       \directlua{Babel.end_process_input()}}%
5102   \fi}%
5103 \AddBabelHook{luatex}{stopcommands}{%
5104   \luabbl@stop
5105   \let\luabbl@stop\relax}
5106 \AddBabelHook{luatex}{patterns}{%
5107   \@ifundefined{bbl@hyphendata@\the\language}%
5108     {\def\bbl@elt##1##2##3##4{%
5109        \ifnum##2=\csname l@##2\endcsname % #2=spanish, dutch:OT1...
5110          \def\bbl@tempb{##3}%
5111          \ifx\bbl@tempb\@empty\else % if not a synonymous
5112            \def\bbl@tempc{{##3}{##4}}%
```

```
5113            \fi
5114            \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5115          \fi}%
5116        \bbl@languages
5117        \@ifundefined{bbl@hyphendata@\the\language}%
5118          {\bbl@info{No hyphenation patterns were set for\\%
5119                    language '#2'. Reported}}%
5120          {\expandafter\expandafter\expandafter\bbl@luapatterns
5121            \csname bbl@hyphendata@\the\language\endcsname}}{}%
5122      \@ifundefined{bbl@patterns@}{}{%
5123        \begingroup
5124          \bbl@xin@{,\number\language,}{,\bbl@pttnlist}%
5125          \ifin@\else
5126            \ifx\bbl@patterns@\@empty\else
5127              \directlua{ Babel.addpatterns(
5128                [[\bbl@patterns@]], \number\language) }%
5129            \fi
5130            \@ifundefined{bbl@patterns@#1}%
5131              \@empty
5132              {\directlua{ Babel.addpatterns(
5133                    [[\space\csname bbl@patterns@#1\endcsname]],
5134                    \number\language) }}%
5135            \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
5136          \fi
5137        \endgroup}%
5138      \bbl@exp{%
5139        \bbl@ifunset{bbl@prehc@\languagename}{}%
5140          {\\\bbl@ifblank{\bbl@cs{prehc@\languagename}}{}%
5141            {\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.

```
5142 \@onlypreamble\babelpatterns
5143 \AtEndOfPackage{%
5144   \newcommand\babelpatterns[2][\@empty]{%
5145     \ifx\bbl@patterns@\relax
5146       \let\bbl@patterns@\@empty
5147     \fi
5148     \ifx\bbl@pttnlist\@empty\else
5149       \bbl@warning{%
5150         You must not intermingle \string\selectlanguage\space and\\%
5151         \string\babelpatterns\space or some patterns will not\\%
5152         be taken into account. Reported}%
5153     \fi
5154     \ifx\@empty#1%
5155       \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
5156     \else
5157       \edef\bbl@tempb{\zap@space#1 \@empty}%
5158       \bbl@for\bbl@tempa\bbl@tempb{%
5159         \bbl@fixname\bbl@tempa
5160         \bbl@iflanguage\bbl@tempa{%
5161           \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
5162             \@ifundefined{bbl@patterns@\bbl@tempa}%
5163               \@empty
5164               {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
5165             #2}}}%
5166     \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.

```
5167 % TODO - to a lua file
5168 \directlua{
5169   Babel = Babel or {}
5170   Babel.linebreaking = Babel.linebreaking or {}
5171   Babel.linebreaking.before = {}
5172   Babel.linebreaking.after = {}
5173   Babel.locale = {} % Free to use, indexed by \localeid
5174   function Babel.linebreaking.add_before(func)
5175     tex.print([[\noexpand\csname bbl@luahyphenate\endcsname]])
5176     table.insert(Babel.linebreaking.before, func)
5177   end
5178   function Babel.linebreaking.add_after(func)
5179     tex.print([[\noexpand\csname bbl@luahyphenate\endcsname]])
5180     table.insert(Babel.linebreaking.after, func)
5181   end
5182 }
5183 \def\bbl@intraspace#1 #2 #3\@@{%
5184   \directlua{
5185     Babel = Babel or {}
5186     Babel.intraspaces = Babel.intraspaces or {}
5187     Babel.intraspaces['\csname bbl@sbcp@\languagename\endcsname'] = %
5188       {b = #1, p = #2, m = #3}
5189     Babel.locale_props[\the\localeid].intraspace = %
5190       {b = #1, p = #2, m = #3}
5191  }}
5192 \def\bbl@intrapenalty#1\@@{%
5193   \directlua{
5194     Babel = Babel or {}
5195     Babel.intrapenalties = Babel.intrapenalties or {}
5196     Babel.intrapenalties['\csname bbl@sbcp@\languagename\endcsname'] = #1
5197     Babel.locale_props[\the\localeid].intrapenalty = #1
5198  }}
5199 \begingroup
5200 \catcode`\%=12
5201 \catcode`\^=14
5202 \catcode`\'=12
5203 \catcode`\~=12
5204 \gdef\bbl@seaintraspace{^
5205   \let\bbl@seaintraspace\relax
5206   \directlua{
5207     Babel = Babel or {}
5208     Babel.sea_enabled = true
5209     Babel.sea_ranges = Babel.sea_ranges or {}
5210     function Babel.set_chranges (script, chrng)
5211       local c = 0
5212       for s, e in string.gmatch(chrng..' ', '(.-)%.%.(.-)%s') do
5213         Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
5214         c = c + 1
5215       end
5216     end
5217     function Babel.sea_disc_to_space (head)
5218       local sea_ranges = Babel.sea_ranges
5219       local last_char = nil
5220       local quad = 655360       ^% 10 pt = 655360 = 10 * 65536
5221       for item in node.traverse(head) do
5222         local i = item.id
5223         if i == node.id'glyph' then
5224           last_char = item
5225         elseif i == 7 and item.subtype == 3 and last_char
5226             and last_char.char > 0x0C99 then
```

```
5227            quad = font.getfont(last_char.font).size
5228            for lg, rg in pairs(sea_ranges) do
5229              if last_char.char > rg[1] and last_char.char < rg[2] then
5230                lg = lg:sub(1, 4)  ^% Remove trailing number of, eg, Cyrl1
5231                local intraspace = Babel.intraspaces[lg]
5232                local intrapenalty = Babel.intrapenalties[lg]
5233                local n
5234                if intrapenalty ~= 0 then
5235                  n = node.new(14, 0)      ^% penalty
5236                  n.penalty = intrapenalty
5237                  node.insert_before(head, item, n)
5238                end
5239                n = node.new(12, 13)      ^% (glue, spaceskip)
5240                node.setglue(n, intraspace.b * quad,
5241                              intraspace.p * quad,
5242                              intraspace.m * quad)
5243                node.insert_before(head, item, n)
5244                node.remove(head, item)
5245              end
5246            end
5247          end
5248        end
5249      end
5250    }^^
5251    \bbl@luahyphenate}
```

12.5   CJK line breaking

Minimal line breaking for CJK scripts, mainly intended for simple documents and short texts as a
secundary 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.

```
5252 \catcode`\%=14
5253 \gdef\bbl@cjkintraspace{%
5254   \let\bbl@cjkintraspace\relax
5255   \directlua{
5256     Babel = Babel or {}
5257     require('babel-data-cjk.lua')
5258     Babel.cjk_enabled = true
5259     function Babel.cjk_linebreak(head)
5260       local GLYPH = node.id'glyph'
5261       local last_char = nil
5262       local quad = 655360      % 10 pt = 655360 = 10 * 65536
5263       local last_class = nil
5264       local last_lang = nil
5265
5266       for item in node.traverse(head) do
5267         if item.id == GLYPH then
5268
5269           local lang = item.lang
5270
5271           local LOCALE = node.get_attribute(item,
5272                 Babel.attr_locale)
5273           local props = Babel.locale_props[LOCALE]
5274
5275           local class = Babel.cjk_class[item.char].c
5276
5277           if props.cjk_quotes and props.cjk_quotes[item.char] then
5278             class = props.cjk_quotes[item.char]
5279           end
5280
```

```
5281            if class == 'cp' then class = 'cl' end % )] as CL
5282            if class == 'id' then class = 'I' end
5283
5284            local br = 0
5285            if class and last_class and Babel.cjk_breaks[last_class][class] then
5286              br = Babel.cjk_breaks[last_class][class]
5287            end
5288
5289            if br == 1 and props.linebreak == 'c' and
5290                lang ~= \the\l@nohyphenation\space and
5291                last_lang ~= \the\l@nohyphenation then
5292              local intrapenalty = props.intrapenalty
5293              if intrapenalty ~= 0 then
5294                local n = node.new(14, 0)      % penalty
5295                n.penalty = intrapenalty
5296                node.insert_before(head, item, n)
5297              end
5298              local intraspace = props.intraspace
5299              local n = node.new(12, 13)       % (glue, spaceskip)
5300              node.setglue(n, intraspace.b * quad,
5301                              intraspace.p * quad,
5302                              intraspace.m * quad)
5303              node.insert_before(head, item, n)
5304            end
5305
5306            if font.getfont(item.font) then
5307              quad = font.getfont(item.font).size
5308            end
5309            last_class = class
5310            last_lang = lang
5311          else % if penalty, glue or anything else
5312            last_class = nil
5313          end
5314        end
5315      lang.hyphenate(head)
5316    end
5317  }%
5318  \bbl@luahyphenate}
5319 \gdef\bbl@luahyphenate{%
5320  \let\bbl@luahyphenate\relax
5321  \directlua{
5322    luatexbase.add_to_callback('hyphenate',
5323    function (head, tail)
5324      if Babel.linebreaking.before then
5325        for k, func in ipairs(Babel.linebreaking.before)  do
5326          func(head)
5327        end
5328      end
5329      if Babel.cjk_enabled then
5330        Babel.cjk_linebreak(head)
5331      end
5332      lang.hyphenate(head)
5333      if Babel.linebreaking.after then
5334        for k, func in ipairs(Babel.linebreaking.after)  do
5335          func(head)
5336        end
5337      end
5338      if Babel.sea_enabled then
5339        Babel.sea_disc_to_space(head)
5340      end
5341    end,
5342    'Babel.hyphenate')
5343  }
```

```
5344 }
5345 \endgroup
5346 \def\bbl@provide@intraspace{%
5347   \bbl@ifunset{bbl@intsp@\languagename}{}%
5348     {\expandafter\ifx\csname bbl@intsp@\languagename\endcsname\@empty\else
5349       \bbl@xin@{/c}{/\bbl@cl{lnbrk}}%
5350       \ifin@           % cjk
5351         \bbl@cjkintraspace
5352         \directlua{
5353           Babel = Babel or {}
5354           Babel.locale_props = Babel.locale_props or {}
5355           Babel.locale_props[\the\localeid].linebreak = 'c'
5356         }%
5357         \bbl@exp{\\\bbl@intraspace\bbl@cl{intsp}\\\@@}%
5358         \ifx\bbl@KVP@intrapenalty\@nnil
5359           \bbl@intrapenalty0\@@
5360         \fi
5361       \else             % sea
5362         \bbl@seaintraspace
5363         \bbl@exp{\\\bbl@intraspace\bbl@cl{intsp}\\\@@}%
5364         \directlua{
5365           Babel = Babel or {}
5366           Babel.sea_ranges = Babel.sea_ranges or {}
5367           Babel.set_chranges('\bbl@cl{sbcp}',
5368                             '\bbl@cl{chrng}')
5369         }%
5370         \ifx\bbl@KVP@intrapenalty\@nnil
5371           \bbl@intrapenalty0\@@
5372         \fi
5373       \fi
5374     \fi
5375     \ifx\bbl@KVP@intrapenalty\@nnil\else
5376       \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
5377     \fi}}
```

12.6 Arabic justification

```
5378 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5379 \def\bblar@chars{%
5380   0628,0629,062A,062B,062C,062D,062E,062F,0630,0631,0632,0633,%
5381   0634,0635,0636,0637,0638,0639,063A,063B,063C,063D,063E,063F,%
5382   0640,0641,0642,0643,0644,0645,0646,0647,0649}
5383 \def\bblar@elongated{%
5384   0626,0628,062A,062B,0633,0634,0635,0636,063B,%
5385   063C,063D,063E,063F,0641,0642,0643,0644,0646,%
5386   0649,064A}
5387 \begingroup
5388   \catcode`\_=11 \catcode`\:=11
5389   \gdef\bblar@nofswarn{\gdef\msg_warning:nnx##1##2##3{}}
5390 \endgroup
5391 \gdef\bbl@arabicjust{%
5392   \let\bbl@arabicjust\relax
5393   \newattribute\bblar@kashida
5394   \directlua{ Babel.attr_kashida = luatexbase.registernumber'bblar@kashida' }%
5395   \bblar@kashida=\z@
5396   \bbl@patchfont{{\bbl@parsejalt}}%
5397   \directlua{
5398     Babel.arabic.elong_map   = Babel.arabic.elong_map or {}
5399     Babel.arabic.elong_map[\the\localeid]   = {}
5400     luatexbase.add_to_callback('post_linebreak_filter',
5401       Babel.arabic.justify, 'Babel.arabic.justify')
5402     luatexbase.add_to_callback('hpack_filter',
5403       Babel.arabic.justify_hbox, 'Babel.arabic.justify_hbox')
```

```
5404    }}%
5405 % Save both node lists to make replacement. TODO. Save also widths to
5406 % make computations
5407 \def\bblar@fetchjalt#1#2#3#4{%
5408    \bbl@exp{\\\bbl@foreach{#1}}{%
5409      \bbl@ifunset{bblar@JE@##1}%
5410        {\setbox\z@\hbox{^^^^200d\char"##1#2}}%
5411        {\setbox\z@\hbox{^^^^200d\char"\@nameuse{bblar@JE@##1}#2}}%
5412      \directlua{%
5413        local last = nil
5414        for item in node.traverse(tex.box[0].head) do
5415          if item.id == node.id'glyph' and item.char > 0x600 and
5416             not (item.char == 0x200D) then
5417            last = item
5418          end
5419        end
5420        Babel.arabic.#3['##1#4'] = last.char
5421      }}}
5422 % Brute force. No rules at all, yet. The ideal: look at jalt table. And
5423 % perhaps other tables (falt?, cswh?). What about kaf? And diacritic
5424 % positioning?
5425 \gdef\bbl@parsejalt{%
5426    \ifx\addfontfeature\@undefined\else
5427      \bbl@xin@{/e}{/\bbl@cl{lnbrk}}%
5428      \ifin@
5429        \directlua{%
5430          if Babel.arabic.elong_map[\the\localeid][\fontid\font] == nil then
5431            Babel.arabic.elong_map[\the\localeid][\fontid\font] = {}
5432            tex.print([[\string\csname\space bbl@parsejalti\endcsname]])
5433          end
5434        }%
5435      \fi
5436    \fi}
5437 \gdef\bbl@parsejalti{%
5438    \begingroup
5439      \let\bbl@parsejalt\relax      % To avoid infinite loop
5440      \edef\bbl@tempb{\fontid\font}%
5441      \bblar@nofswarn
5442      \bblar@fetchjalt\bblar@elongated{}{from}{}%
5443      \bblar@fetchjalt\bblar@chars{^^^^064a}{from}{a}% Alef maksura
5444      \bblar@fetchjalt\bblar@chars{^^^^0649}{from}{y}% Yeh
5445      \addfontfeature{RawFeature=+jalt}%
5446    % \@namedef{bblar@JE@0643}{06AA}% todo: catch medial kaf
5447      \bblar@fetchjalt\bblar@elongated{}{dest}{}%
5448      \bblar@fetchjalt\bblar@chars{^^^^064a}{dest}{a}%
5449      \bblar@fetchjalt\bblar@chars{^^^^0649}{dest}{y}%
5450        \directlua{%
5451          for k, v in pairs(Babel.arabic.from) do
5452            if Babel.arabic.dest[k] and
5453               not (Babel.arabic.from[k] == Babel.arabic.dest[k]) then
5454              Babel.arabic.elong_map[\the\localeid][\bbl@tempb]
5455                [Babel.arabic.from[k]] = Babel.arabic.dest[k]
5456            end
5457          end
5458        }%
5459    \endgroup}
5460 %
5461 \begingroup
5462 \catcode`#=11
5463 \catcode`~=11
5464 \directlua{
5465
5466 Babel.arabic = Babel.arabic or {}
```

```
5467 Babel.arabic.from = {}
5468 Babel.arabic.dest = {}
5469 Babel.arabic.justify_factor = 0.95
5470 Babel.arabic.justify_enabled = true
5471
5472 function Babel.arabic.justify(head)
5473   if not Babel.arabic.justify_enabled then return head end
5474   for line in node.traverse_id(node.id'hlist', head) do
5475     Babel.arabic.justify_hlist(head, line)
5476   end
5477   return head
5478 end
5479
5480 function Babel.arabic.justify_hbox(head, gc, size, pack)
5481   local has_inf = false
5482   if Babel.arabic.justify_enabled and pack == 'exactly' then
5483     for n in node.traverse_id(12, head) do
5484       if n.stretch_order > 0 then has_inf = true end
5485     end
5486     if not has_inf then
5487       Babel.arabic.justify_hlist(head, nil, gc, size, pack)
5488     end
5489   end
5490   return head
5491 end
5492
5493 function Babel.arabic.justify_hlist(head, line, gc, size, pack)
5494   local d, new
5495   local k_list, k_item, pos_inline
5496   local width, width_new, full, k_curr, wt_pos, goal, shift
5497   local subst_done = false
5498   local elong_map = Babel.arabic.elong_map
5499   local last_line
5500   local GLYPH = node.id'glyph'
5501   local KASHIDA = Babel.attr_kashida
5502   local LOCALE = Babel.attr_locale
5503
5504   if line == nil then
5505     line = {}
5506     line.glue_sign = 1
5507     line.glue_order = 0
5508     line.head = head
5509     line.shift = 0
5510     line.width = size
5511   end
5512
5513   % Exclude last line. todo. But-- it discards one-word lines, too!
5514   % ? Look for glue = 12:15
5515   if (line.glue_sign == 1 and line.glue_order == 0) then
5516     elongs = {}     % Stores elongated candidates of each line
5517     k_list = {}     % And all letters with kashida
5518     pos_inline = 0  % Not yet used
5519
5520     for n in node.traverse_id(GLYPH, line.head) do
5521       pos_inline = pos_inline + 1 % To find where it is. Not used.
5522
5523       % Elongated glyphs
5524       if elong_map then
5525         local locale = node.get_attribute(n, LOCALE)
5526         if elong_map[locale] and elong_map[locale][n.font] and
5527             elong_map[locale][n.font][n.char] then
5528           table.insert(elongs, {node = n, locale = locale} )
5529           node.set_attribute(n.prev, KASHIDA, 0)
```

```
5530              end
5531          end
5532
5533        % Tatwil
5534        if Babel.kashida_wts then
5535          local k_wt = node.get_attribute(n, KASHIDA)
5536          if k_wt > 0 then % todo. parameter for multi inserts
5537            table.insert(k_list, {node = n, weight = k_wt, pos = pos_inline})
5538          end
5539        end
5540
5541      end % of node.traverse_id
5542
5543      if #elongs == 0 and #k_list == 0 then goto next_line end
5544      full  = line.width
5545      shift = line.shift
5546      goal  = full * Babel.arabic.justify_factor % A bit crude
5547      width = node.dimensions(line.head)    % The 'natural' width
5548
5549      % == Elongated ==
5550      % Original idea taken from 'chikenize'
5551      while (#elongs > 0 and width < goal) do
5552        subst_done = true
5553        local x = #elongs
5554        local curr = elongs[x].node
5555        local oldchar = curr.char
5556        curr.char = elong_map[elongs[x].locale][curr.font][curr.char]
5557        width = node.dimensions(line.head)  % Check if the line is too wide
5558        % Substitute back if the line would be too wide and break:
5559        if width > goal then
5560          curr.char = oldchar
5561          break
5562        end
5563        % If continue, pop the just substituted node from the list:
5564        table.remove(elongs, x)
5565      end
5566
5567      % == Tatwil ==
5568      if #k_list == 0 then goto next_line end
5569
5570      width = node.dimensions(line.head)    % The 'natural' width
5571      k_curr = #k_list
5572      wt_pos = 1
5573
5574      while width < goal do
5575        subst_done = true
5576        k_item = k_list[k_curr].node
5577        if k_list[k_curr].weight == Babel.kashida_wts[wt_pos] then
5578          d = node.copy(k_item)
5579          d.char = 0x0640
5580          line.head, new = node.insert_after(line.head, k_item, d)
5581          width_new = node.dimensions(line.head)
5582          if width > goal or width == width_new then
5583            node.remove(line.head, new) % Better compute before
5584            break
5585          end
5586          width = width_new
5587        end
5588        if k_curr == 1 then
5589          k_curr = #k_list
5590          wt_pos = (wt_pos >= table.getn(Babel.kashida_wts)) and 1 or wt_pos+1
5591        else
5592          k_curr = k_curr - 1
```

173

```
5593        end
5594      end
5595
5596      ::next_line::
5597
5598      % Must take into account marks and ins, see luatex manual.
5599      % Have to be executed only if there are changes. Investigate
5600      % what's going on exactly.
5601      if subst_done and not gc then
5602        d = node.hpack(line.head, full, 'exactly')
5603        d.shift = shift
5604        node.insert_before(head, line, d)
5605        node.remove(head, line)
5606      end
5607    end % if process line
5608 end
5609 }
5610 \endgroup
5611 \fi\fi % Arabic just block
```

12.7   Common stuff

```
5612 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
5613 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
5614 \DisableBabelHook{babel-fontspec}
5615 ⟨⟨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.

```
5616 % TODO - to a lua file
5617 \directlua{
5618 Babel.script_blocks = {
5619   ['dflt'] = {},
5620   ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
5621               {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
5622   ['Armn'] = {{0x0530, 0x058F}},
5623   ['Beng'] = {{0x0980, 0x09FF}},
5624   ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
5625   ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
5626   ['Cyrl'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
5627               {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
5628   ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
5629   ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
5630               {0xAB00, 0xAB2F}},
5631   ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
5632 % Don't follow strictly Unicode, which places some Coptic letters in
5633 % the 'Greek and Coptic' block
5634   ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
5635   ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
5636               {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
5637               {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
5638               {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
5639               {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
5640               {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
5641   ['Hebr'] = {{0x0590, 0x05FF}},
5642   ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
5643               {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
```

```
5644  ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
5645  ['Knda'] = {{0x0C80, 0x0CFF}},
5646  ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
5647            {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
5648            {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
5649  ['Laoo'] = {{0x0E80, 0x0EFF}},
5650  ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
5651            {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
5652            {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
5653  ['Mahj'] = {{0x11150, 0x1117F}},
5654  ['Mlym'] = {{0x0D00, 0x0D7F}},
5655  ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
5656  ['Orya'] = {{0x0B00, 0x0B7F}},
5657  ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
5658  ['Syrc'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
5659  ['Taml'] = {{0x0B80, 0x0BFF}},
5660  ['Telu'] = {{0x0C00, 0x0C7F}},
5661  ['Tfng'] = {{0x2D30, 0x2D7F}},
5662  ['Thai'] = {{0x0E00, 0x0E7F}},
5663  ['Tibt'] = {{0x0F00, 0x0FFF}},
5664  ['Vaii'] = {{0xA500, 0xA63F}},
5665  ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
5666 }
5667
5668 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
5669 Babel.script_blocks.Hant = Babel.script_blocks.Hans
5670 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
5671
5672 function Babel.locale_map(head)
5673  if not Babel.locale_mapped then return head end
5674
5675  local LOCALE = Babel.attr_locale
5676  local GLYPH = node.id('glyph')
5677  local inmath = false
5678  local toloc_save
5679  for item in node.traverse(head) do
5680    local toloc
5681    if not inmath and item.id == GLYPH then
5682      % Optimization: build a table with the chars found
5683      if Babel.chr_to_loc[item.char] then
5684        toloc = Babel.chr_to_loc[item.char]
5685      else
5686        for lc, maps in pairs(Babel.loc_to_scr) do
5687          for _, rg in pairs(maps) do
5688            if item.char >= rg[1] and item.char <= rg[2] then
5689              Babel.chr_to_loc[item.char] = lc
5690              toloc = lc
5691              break
5692            end
5693          end
5694        end
5695      end
5696      % Now, take action, but treat composite chars in a different
5697      % fashion, because they 'inherit' the previous locale. Not yet
5698      % optimized.
5699      if not toloc and
5700          (item.char >= 0x0300 and item.char <= 0x036F) or
5701          (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
5702          (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
5703        toloc = toloc_save
5704      end
5705      if toloc and toloc > -1 then
5706        if Babel.locale_props[toloc].lg then
```

```
5707          item.lang = Babel.locale_props[toloc].lg
5708          node.set_attribute(item, LOCALE, toloc)
5709        end
5710        if Babel.locale_props[toloc]['/'..item.font] then
5711          item.font = Babel.locale_props[toloc]['/'..item.font]
5712        end
5713        toloc_save = toloc
5714      end
5715    elseif not inmath and item.id == 7 then
5716      item.replace = item.replace and Babel.locale_map(item.replace)
5717      item.pre     = item.pre and Babel.locale_map(item.pre)
5718      item.post    = item.post and Babel.locale_map(item.post)
5719    elseif item.id == node.id'math' then
5720      inmath = (item.subtype == 0)
5721    end
5722  end
5723  return head
5724 end
5725 }
```

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

```
5726 \newcommand\babelcharproperty[1]{%
5727   \count@=#1\relax
5728   \ifvmode
5729     \expandafter\bbl@chprop
5730   \else
5731     \bbl@error{\string\babelcharproperty\space can be used only in\\%
5732                vertical mode (preamble or between paragraphs)}%
5733                {See the manual for futher info}%
5734   \fi}
5735 \newcommand\bbl@chprop[3][\the\count@]{%
5736   \@tempcnta=#1\relax
5737   \bbl@ifunset{bbl@chprop@#2}%
5738     {\bbl@error{No property named '#2'. Allowed values are\\%
5739                direction (bc), mirror (bmg), and linebreak (lb)}%
5740                {See the manual for futher info}}%
5741     {}%
5742   \loop
5743     \bbl@cs{chprop@#2}{#3}%
5744   \ifnum\count@<\@tempcnta
5745     \advance\count@\@ne
5746   \repeat}
5747 \def\bbl@chprop@direction#1{%
5748   \directlua{
5749     Babel.characters[\the\count@] =  Babel.characters[\the\count@] or {}
5750     Babel.characters[\the\count@]['d'] = '#1'
5751   }}
5752 \let\bbl@chprop@bc\bbl@chprop@direction
5753 \def\bbl@chprop@mirror#1{%
5754   \directlua{
5755     Babel.characters[\the\count@] =  Babel.characters[\the\count@] or {}
5756     Babel.characters[\the\count@]['m'] = '\number#1'
5757   }}
5758 \let\bbl@chprop@bmg\bbl@chprop@mirror
5759 \def\bbl@chprop@linebreak#1{%
5760   \directlua{
5761     Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
5762     Babel.cjk_characters[\the\count@]['c'] = '#1'
5763   }}
5764 \let\bbl@chprop@lb\bbl@chprop@linebreak
5765 \def\bbl@chprop@locale#1{%
5766   \directlua{
```

```
5767    Babel.chr_to_loc = Babel.chr_to_loc or {}
5768    Babel.chr_to_loc[\the\count@] =
5769      \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@@#1}}\space
5770  }}
```

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.

```
5771 \directlua{
5772    Babel.nohyphenation = \the\l@nohyphenation
5773 }
```

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).

```
5774 \begingroup
5775 \catcode`\~=12
5776 \catcode`\%=12
5777 \catcode`\&=14
5778 \catcode`\|=12
5779 \gdef\babelprehyphenation{&%
5780    \@ifnextchar[{\bbl@settransform{0}}{\bbl@settransform{0}[]}}
5781 \gdef\babelposthyphenation{&%
5782    \@ifnextchar[{\bbl@settransform{1}}{\bbl@settransform{1}[]}}
5783 \gdef\bbl@postlinebreak{\bbl@settransform{2}[]} &% WIP
5784 \gdef\bbl@settransform#1[#2]#3#4#5{&%
5785    \ifcase#1
5786      \bbl@activateprehyphen
5787    \or
5788      \bbl@activateposthyphen
5789    \fi
5790    \begingroup
5791      \def\babeltempa{\bbl@add@list\babeltempb}&%
5792      \let\babeltempb\@empty
5793      \def\bbl@tempa{#5}&%
5794      \bbl@replace\bbl@tempa{,}{ ,}&% TODO. Ugly trick to preserve {}
5795      \expandafter\bbl@foreach\expandafter{\bbl@tempa}{&%
5796        \bbl@ifsamestring{##1}{remove}&%
5797          {\bbl@add@list\babeltempb{nil}}&%
5798          {\directlua{
5799             local rep = [=[##1]=]
5800             rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
5801             rep = rep:gsub('^%s*(insert)%s*,', 'insert = true, ')
5802             rep = rep:gsub('(string)%s*=%s*([^%s,]*)', Babel.capture_func)
5803             if #1 == 0 or #1 == 2 then
5804               rep = rep:gsub('(space)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
5805                 'space = {' .. '%2, %3, %4' .. '}')
5806               rep = rep:gsub('(spacefactor)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
5807                 'spacefactor = {' .. '%2, %3, %4' .. '}')
5808               rep = rep:gsub('(kashida)%s*=%s*([^%s,]*)', Babel.capture_kashida)
5809             else
5810               rep = rep:gsub(   '(no)%s*=%s*([^%s,]*)', Babel.capture_func)
5811               rep = rep:gsub(  '(pre)%s*=%s*([^%s,]*)', Babel.capture_func)
5812               rep = rep:gsub( '(post)%s*=%s*([^%s,]*)', Babel.capture_func)
5813             end
5814             tex.print([[\string\babeltempa{{]] .. rep .. [[}}]])
5815          }}&%
5816      \bbl@foreach\babeltempb{&%
5817        \bbl@forkv{{##1}}{&%
```

```
5818        \in@{,####1,}{,nil,step,data,remove,insert,string,no,pre,&%
5819            no,post,penalty,kashida,space,spacefactor,}&%
5820        \ifin@\else
5821          \bbl@error
5822            {Bad option '####1' in a transform.\\&%
5823             I'll ignore it but expect more errors}&%
5824            {See the manual for further info.}&%
5825        \fi}}&%
5826    \let\bbl@kv@attribute\relax
5827    \let\bbl@kv@label\relax
5828    \bbl@forkv{#2}{\bbl@csarg\edef{kv@##1}{##2}}&%
5829    \ifx\bbl@kv@attribute\relax\else
5830      \edef\bbl@kv@attribute{\expandafter\bbl@stripslash\bbl@kv@attribute}&%
5831    \fi
5832    \directlua{
5833      local lbkr = Babel.linebreaking.replacements[#1]
5834      local u = unicode.utf8
5835      local id, attr, label
5836      if #1 == 0 or #1 == 2 then
5837        id = \the\csname bbl@id@@#3\endcsname\space
5838      else
5839        id = \the\csname l@#3\endcsname\space
5840      end
5841      \ifx\bbl@kv@attribute\relax
5842        attr = -1
5843      \else
5844        attr = luatexbase.registernumber'\bbl@kv@attribute'
5845      \fi
5846      \ifx\bbl@kv@label\relax\else  &% Same refs:
5847        label = [==[\bbl@kv@label]==]
5848      \fi
5849      &% Convert pattern:
5850      local patt = string.gsub([==[#4]==], '%s', '')
5851      if #1 == 0 or #1 == 2 then
5852        patt = string.gsub(patt, '|', ' ')
5853      end
5854      if not u.find(patt, '()', nil, true) then
5855        patt = '()' .. patt .. '()'
5856      end
5857      if #1 == 1 then
5858        patt = string.gsub(patt, '%(%)%^', '^()')
5859        patt = string.gsub(patt, '%$%(%)', '()$')
5860      end
5861      patt = u.gsub(patt, '{(.)}',
5862              function (n)
5863                return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
5864              end)
5865      patt = u.gsub(patt, '{(%x%x%x%x+)}',
5866              function (n)
5867                return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%%1')
5868              end)
5869      lbkr[id] = lbkr[id] or {}
5870      table.insert(lbkr[id],
5871        { label=label, attr=attr, pattern=patt, replace={\babeltempb} })
5872    }&%
5873  \endgroup}
5874 \endgroup
5875 \def\bbl@activateposthyphen{%
5876  \let\bbl@activateposthyphen\relax
5877  \directlua{
5878    require('babel-transforms.lua')
5879    Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
5880  }}
```

```
5881 \def\bbl@activateprehyphen{%
5882   \let\bbl@activateprehyphen\relax
5883   \directlua{
5884     require('babel-transforms.lua')
5885     Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
5886   }}
```

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.

```
5887 \def\bbl@activate@preotf{%
5888   \let\bbl@activate@preotf\relax  % only once
5889   \directlua{
5890     Babel = Babel or {}
5891     %
5892     function Babel.pre_otfload_v(head)
5893       if Babel.numbers and Babel.digits_mapped then
5894         head = Babel.numbers(head)
5895       end
5896       if Babel.bidi_enabled then
5897         head = Babel.bidi(head, false, dir)
5898       end
5899       return head
5900     end
5901     %
5902     function Babel.pre_otfload_h(head, gc, sz, pt, dir)
5903       if Babel.numbers and Babel.digits_mapped then
5904         head = Babel.numbers(head)
5905       end
5906       if Babel.bidi_enabled then
5907         head = Babel.bidi(head, false, dir)
5908       end
5909       return head
5910     end
5911     %
5912     luatexbase.add_to_callback('pre_linebreak_filter',
5913       Babel.pre_otfload_v,
5914       'Babel.pre_otfload_v',
5915       luatexbase.priority_in_callback('pre_linebreak_filter',
5916         'luaotfload.node_processor') or nil)
5917     %
5918     luatexbase.add_to_callback('hpack_filter',
5919       Babel.pre_otfload_h,
5920       'Babel.pre_otfload_h',
5921       luatexbase.priority_in_callback('hpack_filter',
5922         'luaotfload.node_processor') or nil)
5923   }}
```

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=.

```
5924 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5925   \let\bbl@beforeforeign\leavevmode
5926   \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
5927   \RequirePackage{luatexbase}
5928   \bbl@activate@preotf
5929   \directlua{
5930     require('babel-data-bidi.lua')
5931     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
5932       require('babel-bidi-basic.lua')
5933     \or
```

```
5934        require('babel-bidi-basic-r.lua')
5935      \fi}
5936    % TODO - to locale_props, not as separate attribute
5937    \newattribute\bbl@attr@dir
5938    \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
5939    % TODO. I don't like it, hackish:
5940    \bbl@exp{\output{\bodydir\pagedir\the\output}}
5941    \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
5942  \fi\fi
5943  \chardef\bbl@thetextdir\z@
5944  \chardef\bbl@thepardir\z@
5945  \def\bbl@getluadir#1{%
5946    \directlua{
5947      if tex.#1dir == 'TLT' then
5948        tex.sprint('0')
5949      elseif tex.#1dir == 'TRT' then
5950        tex.sprint('1')
5951      end}}
5952  \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
5953    \ifcase#3\relax
5954      \ifcase\bbl@getluadir{#1}\relax\else
5955        #2 TLT\relax
5956      \fi
5957    \else
5958      \ifcase\bbl@getluadir{#1}\relax
5959        #2 TRT\relax
5960      \fi
5961    \fi}
5962  \def\bbl@thedir{0}
5963  \def\bbl@textdir#1{%
5964    \bbl@setluadir{text}\textdir{#1}%
5965    \chardef\bbl@thetextdir#1\relax
5966    \edef\bbl@thedir{\the\numexpr\bbl@thepardir*3+#1}%
5967    \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*3+#1}}
5968  \def\bbl@pardir#1{%
5969    \bbl@setluadir{par}\pardir{#1}%
5970    \chardef\bbl@thepardir#1\relax}
5971  \def\bbl@bodydir{\bbl@setluadir{body}\bodydir}
5972  \def\bbl@pagedir{\bbl@setluadir{page}\pagedir}
5973  \def\bbl@dirparastext{\pardir\the\textdir\relax}%    %%%%
5974  %
5975  \ifnum\bbl@bidimode>\z@
5976    \def\bbl@insidemath{0}%
5977    \def\bbl@everymath{\def\bbl@insidemath{1}}
5978    \def\bbl@everydisplay{\def\bbl@insidemath{2}}
5979    \frozen@everymath\expandafter{%
5980      \expandafter\bbl@everymath\the\frozen@everymath}
5981    \frozen@everydisplay\expandafter{%
5982      \expandafter\bbl@everydisplay\the\frozen@everydisplay}
5983    \AtBeginDocument{
5984      \directlua{
5985        function Babel.math_box_dir(head)
5986          if not (token.get_macro('bbl@insidemath') == '0') then
5987            if Babel.hlist_has_bidi(head) then
5988              local d = node.new(node.id'dir')
5989              d.dir = '+TRT'
5990              node.insert_before(head, node.has_glyph(head), d)
5991              for item in node.traverse(head) do
5992                node.set_attribute(item,
5993                  Babel.attr_dir, token.get_macro('bbl@thedir'))
5994              end
5995            end
5996          end
```

```
5997        return head
5998      end
5999      luatexbase.add_to_callback("hpack_filter", Babel.math_box_dir,
6000        "Babel.math_box_dir", 0)
6001  }}%
6002 \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.

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

```
6045        \chardef\bbl@thetextdir\z@
6046        \bbl@add\normalfont{\bbl@eqnodir}%
6047        \ifcase\bbl@eqnpos
6048          \let\bbl@puteqno\bbl@eqno@flip
6049        \or
6050          \let\bbl@puteqno\bbl@leqno@flip
6051        \fi
6052      \fi}%
6053    \ifnum\bbl@eqnpos=\tw@\else
6054      \def\endequation{\bbl@puteqno{\@eqnnum}$$\@ignoretrue}%
6055    \fi
6056    \AddToHook{env/eqnarray/begin}{%
6057      \ifnum\bbl@thetextdir>\z@
6058        \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6059        \chardef\bbl@thetextdir\z@
6060        \bbl@add\normalfont{\bbl@eqnodir}%
6061        \ifnum\bbl@eqnpos=\@ne
6062          \def\@eqnnum{%
6063            \setbox\z@\hbox{\bbl@eqnum}%
6064            \hbox to0.01pt{\hss\hbox to\displaywidth{\box\z@\hss}}}%
6065        \else
6066          \let\@eqnnum\bbl@eqnum
6067        \fi
6068      \fi}
6069    % Hack. YA luatex bug?:
6070    \expandafter\bbl@sreplace\csname] \endcsname{$$}{\eqno\kern.001pt$$}%
6071  \else % amstex
6072    \ifx\bbl@noamsmath\@undefined
6073      \ifnum\bbl@eqnpos=\@ne
6074        \let\bbl@ams@lap\hbox
6075      \else
6076        \let\bbl@ams@lap\llap
6077      \fi
6078      \ExplSyntaxOn
6079      \bbl@sreplace\intertext@{\normalbaselines}%
6080        {\normalbaselines
6081         \ifx\bbl@eqnodir\relax\else\bbl@pardir\@ne\bbl@eqnodir\fi}%
6082      \ExplSyntaxOff
6083      \def\bbl@ams@tagbox#1#2{#1{\bbl@eqnodir#2}}% #1=hbox|@lap|flip
6084      \ifx\bbl@ams@lap\hbox % leqno
6085        \def\bbl@ams@flip#1{%
6086          \hbox to 0.01pt{\hss\hbox to\displaywidth{{#1}\hss}}}%
6087      \else % eqno
6088        \def\bbl@ams@flip#1{%
6089          \hbox to 0.01pt{\hbox to\displaywidth{\hss{#1}}\hss}}%
6090      \fi
6091      \def\bbl@ams@preset#1{%
6092        \ifnum\bbl@thetextdir>\z@
6093          \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6094          \bbl@sreplace\textdef@{\hbox}{\bbl@ams@tagbox\hbox}%
6095          \bbl@sreplace\maketag@@@{\hbox}{\bbl@ams@tagbox#1}%
6096        \fi}%
6097      \ifnum\bbl@eqnpos=\tw@\else
6098        \def\bbl@ams@equation{%
6099          \ifnum\bbl@thetextdir>\z@
6100            \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6101            \chardef\bbl@thetextdir\z@
6102            \bbl@add\normalfont{\bbl@eqnodir}%
6103            \ifcase\bbl@eqnpos
6104              \def\veqno##1##2{\bbl@eqno@flip{##1##2}}%
6105            \or
6106              \def\veqno##1##2{\bbl@leqno@flip{##1##2}}%
6107            \fi
```

182

```
6108            \fi}%
6109          \AddToHook{env/equation/begin}{\bbl@ams@equation}%
6110          \AddToHook{env/equation*/begin}{\bbl@ams@equation}%
6111        \fi
6112        \AddToHook{env/cases/begin}{\bbl@ams@preset\bbl@ams@lap}%
6113        \AddToHook{env/multline/begin}{\bbl@ams@preset\hbox}%
6114        \AddToHook{env/gather/begin}{\bbl@ams@preset\bbl@ams@lap}%
6115        \AddToHook{env/gather*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6116        \AddToHook{env/align/begin}{\bbl@ams@preset\bbl@ams@lap}%
6117        \AddToHook{env/align*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6118        \AddToHook{env/eqnalign/begin}{\bbl@ams@preset\hbox}%
6119        % Hackish, for proper alignment. Don't ask me why it works!:
6120        \bbl@exp{% Avoid a 'visible' conditional
6121          \\\AddToHook{env/align*/end}{\<iftag@>\<else>\\\tag*{}\<fi>}}%
6122        \AddToHook{env/flalign/begin}{\bbl@ams@preset\hbox}%
6123        \AddToHook{env/split/before}{%
6124          \ifnum\bbl@thetextdir>\z@
6125            \bbl@ifsamestring\@currenvir{equation}%
6126              {\ifx\bbl@ams@lap\hbox % leqno
6127                \def\bbl@ams@flip#1{%
6128                  \hbox to 0.01pt{\hbox to\displaywidth{{#1}\hss}\hss}}%
6129              \else
6130                \def\bbl@ams@flip#1{%
6131                  \hbox to 0.01pt{\hss\hbox to\displaywidth{\hss{#1}}}}%
6132              \fi}%
6133              {}%
6134          \fi}%
6135      \fi
6136    \fi}
6137 \fi
6138 \ifx\bbl@opt@layout\@nnil\endinput\fi  % if no layout
6139 \ifnum\bbl@bidimode>\z@
6140  \def\bbl@nextfake#1{%  non-local changes, use always inside a group!
6141    \bbl@exp{%
6142      \def\\\bbl@insidemath{0}%
6143      \mathdir\the\bodydir
6144      #1%                Once entered in math, set boxes to restore values
6145      \<ifmmode>%
6146        \everyvbox{%
6147          \the\everyvbox
6148          \bodydir\the\bodydir
6149          \mathdir\the\mathdir
6150          \everyhbox{\the\everyhbox}%
6151          \everyvbox{\the\everyvbox}}%
6152        \everyhbox{%
6153          \the\everyhbox
6154          \bodydir\the\bodydir
6155          \mathdir\the\mathdir
6156          \everyhbox{\the\everyhbox}%
6157          \everyvbox{\the\everyvbox}}%
6158      \<fi>}}%
6159  \def\@hangfrom#1{%
6160    \setbox\@tempboxa\hbox{{#1}}%
6161    \hangindent\wd\@tempboxa
6162    \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
6163      \shapemode\@ne
6164    \fi
6165    \noindent\box\@tempboxa}
6166 \fi
6167 \IfBabelLayout{tabular}
6168  {\let\bbl@OL@@tabular\@tabular
6169   \bbl@replace\@tabular{$}{\bbl@nextfake$}%
6170   \let\bbl@NL@@tabular\@tabular
```

```
6171    \AtBeginDocument{%
6172      \ifx\bbl@NL@@tabular\@tabular\else
6173        \bbl@replace\@tabular{$}{\bbl@nextfake$}%
6174        \let\bbl@NL@@tabular\@tabular
6175      \fi}}
6176    {}
6177 \IfBabelLayout{lists}
6178    {\let\bbl@OL@list\list
6179     \bbl@sreplace\list{\parshape}{\bbl@listparshape}%
6180     \let\bbl@NL@list\list
6181     \def\bbl@listparshape#1#2#3{%
6182       \parshape #1 #2 #3 %
6183       \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
6184         \shapemode\tw@
6185       \fi}}
6186    {}
6187 \IfBabelLayout{graphics}
6188    {\let\bbl@pictresetdir\relax
6189     \def\bbl@pictsetdir#1{%
6190       \ifcase\bbl@thetextdir
6191         \let\bbl@pictresetdir\relax
6192       \else
6193         \ifcase#1\bodydir TLT  % Remember this sets the inner boxes
6194           \or\textdir TLT
6195           \else\bodydir TLT \textdir TLT
6196         \fi
6197         % \(text|par)dir required in pgf:
6198         \def\bbl@pictresetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}%
6199       \fi}%
6200     \AddToHook{env/picture/begin}{\bbl@pictsetdir\tw@}%
6201     \directlua{
6202       Babel.get_picture_dir = true
6203       Babel.picture_has_bidi = 0
6204       %
6205       function Babel.picture_dir (head)
6206         if not Babel.get_picture_dir then return head end
6207         if Babel.hlist_has_bidi(head) then
6208           Babel.picture_has_bidi = 1
6209         end
6210         return head
6211       end
6212       luatexbase.add_to_callback("hpack_filter", Babel.picture_dir,
6213         "Babel.picture_dir")
6214    }%
6215     \AtBeginDocument{%
6216       \long\def\put(#1,#2)#3{%
6217         \@killglue
6218         % Try:
6219         \ifx\bbl@pictresetdir\relax
6220           \def\bbl@tempc{0}%
6221         \else
6222           \directlua{
6223             Babel.get_picture_dir = true
6224             Babel.picture_has_bidi = 0
6225           }%
6226           \setbox\z@\hb@xt@\z@{%
6227             \@defaultunitsset\@tempdimc{#1}\unitlength
6228             \kern\@tempdimc
6229             #3\hss}% TODO: #3 executed twice (below). That's bad.
6230           \edef\bbl@tempc{\directlua{tex.print(Babel.picture_has_bidi)}}}%
6231         \fi
6232         % Do:
6233         \@defaultunitsset\@tempdimc{#2}\unitlength
```

```
6234        \raise\@tempdimc\hb@xt@\z@{%
6235          \@defaultunitsset\@tempdimc{#1}\unitlength
6236          \kern\@tempdimc
6237          {\ifnum\bbl@tempc>\z@\bbl@pictresetdir\fi#3}\hss}%
6238        \ignorespaces}%
6239      \MakeRobust\put}%
6240    \AtBeginDocument
6241      {\AddToHook{cmd/diagbox@pict/before}{\let\bbl@pictsetdir\@gobble}%
6242       \ifx\pgfpicture\@undefined\else % TODO. Allow deactivate?
6243         \AddToHook{env/pgfpicture/begin}{\bbl@pictsetdir\@ne}%
6244         \bbl@add\pgfinterruptpicture{\bbl@pictresetdir}%
6245         \bbl@add\pgfsys@beginpicture{\bbl@pictsetdir\z@}%
6246       \fi
6247       \ifx\tikzpicture\@undefined\else
6248         \AddToHook{env/tikzpicture/begin}{\bbl@pictsetdir\z@}%
6249         \bbl@add\tikz@atbegin@node{\bbl@pictresetdir}%
6250         \bbl@sreplace\tikz{\begingroup}{\begingroup\bbl@pictsetdir\tw@}%
6251       \fi
6252       \ifx\tcolorbox\@undefined\else
6253         \def\tcb@drawing@env@begin{%
6254         \csname tcb@before@\tcb@split@state\endcsname
6255         \bbl@pictsetdir\tw@
6256         \begin{kvtcb@graphenv}%
6257         \tcb@bbdraw%
6258         \tcb@apply@graph@patches
6259         }%
6260       \def\tcb@drawing@env@end{%
6261       \end{kvtcb@graphenv}%
6262       \bbl@pictresetdir
6263       \csname tcb@after@\tcb@split@state\endcsname
6264       }%
6265     \fi
6266   }}
6267  {}
```

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.

```
6268 \IfBabelLayout{counters}%
6269   {\let\bbl@OL@@textsuperscript\@textsuperscript
6270    \bbl@sreplace\@textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
6271    \let\bbl@latinarabic=\@arabic
6272    \let\bbl@OL@@arabic\@arabic
6273    \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
6274    \@ifpackagewith{babel}{bidi=default}%
6275      {\let\bbl@asciiroman=\@roman
6276       \let\bbl@OL@@roman\@roman
6277       \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciiroman#1}}}%
6278       \let\bbl@asciiRoman=\@Roman
6279       \let\bbl@OL@@roman\@Roman
6280       \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}%
6281       \let\bbl@OL@labelenumii\labelenumii
6282       \def\labelenumii{)\theenumii(}%
6283       \let\bbl@OL@p@enumiii\p@enumiii
6284       \def\p@enumiii{\p@enumii)\theenumii(}}{}}{}
6285 ⟨⟨Footnote changes⟩⟩
6286 \IfBabelLayout{footnotes}%
6287   {\let\bbl@OL@footnote\footnote
6288    \BabelFootnote\footnote\languagename{}{}%
6289    \BabelFootnote\localfootnote\languagename{}{}%
6290    \BabelFootnote\mainfootnote{}{}{}}
6291  {}
```

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.

```
6292 \IfBabelLayout{extras}%
6293   {\let\bbl@OL@underline\underline
6294    \bbl@sreplace\underline{$\@@underline}{\bbl@nextfake$\@@underline}%
6295    \let\bbl@OL@LaTeX2e\LaTeX2e
6296    \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
6297      \if b\expandafter\@car\f@series\@nil\boldmath\fi
6298      \babelsublr{%
6299        \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}}}
6300   {}
6301 ⟨/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.

```
6302 ⟨*transforms⟩
6303 Babel.linebreaking.replacements = {}
6304 Babel.linebreaking.replacements[0] = {}  -- pre
6305 Babel.linebreaking.replacements[1] = {}  -- post
6306 Babel.linebreaking.replacements[2] = {}  -- post-line WIP
6307
6308 -- Discretionaries contain strings as nodes
6309 function Babel.str_to_nodes(fn, matches, base)
6310   local n, head, last
6311   if fn == nil then return nil end
6312   for s in string.utfvalues(fn(matches)) do
6313     if base.id == 7 then
6314       base = base.replace
6315     end
6316     n = node.copy(base)
6317     n.char    = s
6318     if not head then
6319       head = n
6320     else
6321       last.next = n
6322     end
6323     last = n
6324   end
6325   return head
6326 end
6327
6328 Babel.fetch_subtext = {}
6329
6330 Babel.ignore_pre_char = function(node)
6331   return (node.lang == Babel.nohyphenation)
6332 end
6333
6334 -- Merging both functions doesn't seen feasible, because there are too
6335 -- many differences.
6336 Babel.fetch_subtext[0] = function(head)
6337   local word_string = ''
6338   local word_nodes = {}
```

```
6339   local lang
6340   local item = head
6341   local inmath = false
6342
6343   while item do
6344
6345     if item.id == 11 then
6346       inmath = (item.subtype == 0)
6347     end
6348
6349     if inmath then
6350       -- pass
6351
6352     elseif item.id == 29 then
6353       local locale = node.get_attribute(item, Babel.attr_locale)
6354
6355       if lang == locale or lang == nil then
6356         lang = lang or locale
6357         if Babel.ignore_pre_char(item) then
6358           word_string = word_string .. Babel.us_char
6359         else
6360           word_string = word_string .. unicode.utf8.char(item.char)
6361         end
6362         word_nodes[#word_nodes+1] = item
6363       else
6364         break
6365       end
6366
6367     elseif item.id == 12 and item.subtype == 13 then
6368       word_string = word_string .. ' '
6369       word_nodes[#word_nodes+1] = item
6370
6371     -- Ignore leading unrecognized nodes, too.
6372     elseif word_string ~= '' then
6373       word_string = word_string .. Babel.us_char
6374       word_nodes[#word_nodes+1] = item  -- Will be ignored
6375     end
6376
6377     item = item.next
6378   end
6379
6380   -- Here and above we remove some trailing chars but not the
6381   -- corresponding nodes. But they aren't accessed.
6382   if word_string:sub(-1) == ' ' then
6383     word_string = word_string:sub(1,-2)
6384   end
6385   word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
6386   return word_string, word_nodes, item, lang
6387 end
6388
6389 Babel.fetch_subtext[1] = function(head)
6390   local word_string = ''
6391   local word_nodes = {}
6392   local lang
6393   local item = head
6394   local inmath = false
6395
6396   while item do
6397
6398     if item.id == 11 then
6399       inmath = (item.subtype == 0)
6400     end
6401
```

```
6402      if inmath then
6403        -- pass
6404
6405      elseif item.id == 29 then
6406        if item.lang == lang or lang == nil then
6407          if (item.char ~= 124) and (item.char ~= 61) then -- not =, not |
6408            lang = lang or item.lang
6409            word_string = word_string .. unicode.utf8.char(item.char)
6410            word_nodes[#word_nodes+1] = item
6411          end
6412        else
6413          break
6414        end
6415
6416      elseif item.id == 7 and item.subtype == 2 then
6417        word_string = word_string .. '='
6418        word_nodes[#word_nodes+1] = item
6419
6420      elseif item.id == 7 and item.subtype == 3 then
6421        word_string = word_string .. '|'
6422        word_nodes[#word_nodes+1] = item
6423
6424      -- (1) Go to next word if nothing was found, and (2) implicitly
6425      -- remove leading USs.
6426      elseif word_string == '' then
6427        -- pass
6428
6429      -- This is the responsible for splitting by words.
6430      elseif (item.id == 12 and item.subtype == 13) then
6431        break
6432
6433      else
6434        word_string = word_string .. Babel.us_char
6435        word_nodes[#word_nodes+1] = item  -- Will be ignored
6436      end
6437
6438      item = item.next
6439    end
6440
6441    word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
6442    return word_string, word_nodes, item, lang
6443 end
6444
6445 function Babel.pre_hyphenate_replace(head)
6446    Babel.hyphenate_replace(head, 0)
6447 end
6448
6449 function Babel.post_hyphenate_replace(head)
6450    Babel.hyphenate_replace(head, 1)
6451 end
6452
6453 Babel.us_char = string.char(31)
6454
6455 function Babel.hyphenate_replace(head, mode)
6456    local u = unicode.utf8
6457    local lbkr = Babel.linebreaking.replacements[mode]
6458    if mode == 2 then mode = 0 end -- WIP
6459
6460    local word_head = head
6461
6462    while true do  -- for each subtext block
6463
6464      local w, w_nodes, nw, lang = Babel.fetch_subtext[mode](word_head)
```

```
6465
6466    if Babel.debug then
6467      print()
6468      print((mode == 0) and '@@@@<' or '@@@@>', w)
6469    end
6470
6471    if nw == nil and w == '' then break end
6472
6473    if not lang then goto next end
6474    if not lbkr[lang] then goto next end
6475
6476    -- For each saved (pre|post)hyphenation. TODO. Reconsider how
6477    -- loops are nested.
6478    for k=1, #lbkr[lang] do
6479      local p = lbkr[lang][k].pattern
6480      local r = lbkr[lang][k].replace
6481      local attr = lbkr[lang][k].attr or -1
6482
6483      if Babel.debug then
6484        print('*****', p, mode)
6485      end
6486
6487      -- This variable is set in some cases below to the first *byte*
6488      -- after the match, either as found by u.match (faster) or the
6489      -- computed position based on sc if w has changed.
6490      local last_match = 0
6491      local step = 0
6492
6493      -- For every match.
6494      while true do
6495        if Babel.debug then
6496          print('=====')
6497        end
6498        local new  -- used when inserting and removing nodes
6499
6500        local matches = { u.match(w, p, last_match) }
6501
6502        if #matches < 2 then break end
6503
6504        -- Get and remove empty captures (with ()'s, which return a
6505        -- number with the position), and keep actual captures
6506        -- (from (...)), if any, in matches.
6507        local first = table.remove(matches, 1)
6508        local last  = table.remove(matches, #matches)
6509        -- Non re-fetched substrings may contain \31, which separates
6510        -- subsubstrings.
6511        if string.find(w:sub(first, last-1), Babel.us_char) then break end
6512
6513        local save_last = last -- with A()BC()D, points to D
6514
6515        -- Fix offsets, from bytes to unicode. Explained above.
6516        first = u.len(w:sub(1, first-1)) + 1
6517        last  = u.len(w:sub(1, last-1)) -- now last points to C
6518
6519        -- This loop stores in a small table the nodes
6520        -- corresponding to the pattern. Used by 'data' to provide a
6521        -- predictable behavior with 'insert' (w_nodes is modified on
6522        -- the fly), and also access to 'remove'd nodes.
6523        local sc = first-1           -- Used below, too
6524        local data_nodes = {}
6525
6526        local enabled = true
6527        for q = 1, last-first+1 do
```

```
6528           data_nodes[q] = w_nodes[sc+q]
6529           if enabled
6530              and attr > -1
6531              and not node.has_attribute(data_nodes[q], attr)
6532            then
6533              enabled = false
6534          end
6535        end
6536
6537        -- This loop traverses the matched substring and takes the
6538        -- corresponding action stored in the replacement list.
6539        -- sc = the position in substr nodes / string
6540        -- rc = the replacement table index
6541        local rc = 0
6542
6543        while rc < last-first+1 do -- for each replacement
6544          if Babel.debug then
6545            print('.....', rc + 1)
6546          end
6547          sc = sc + 1
6548          rc = rc + 1
6549
6550          if Babel.debug then
6551            Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6552            local ss = ''
6553            for itt in node.traverse(head) do
6554             if itt.id == 29 then
6555               ss = ss .. unicode.utf8.char(itt.char)
6556             else
6557               ss = ss .. '{' .. itt.id .. '}'
6558             end
6559            end
6560            print('*****************', ss)
6561
6562          end
6563
6564          local crep = r[rc]
6565          local item = w_nodes[sc]
6566          local item_base = item
6567          local placeholder = Babel.us_char
6568          local d
6569
6570          if crep and crep.data then
6571            item_base = data_nodes[crep.data]
6572          end
6573
6574          if crep then
6575            step = crep.step or 0
6576          end
6577
6578          if (not enabled) or (crep and next(crep) == nil) then -- = {}
6579            last_match = save_last    -- Optimization
6580            goto next
6581
6582          elseif crep == nil or crep.remove then
6583            node.remove(head, item)
6584            table.remove(w_nodes, sc)
6585            w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
6586            sc = sc - 1  -- Nothing has been inserted.
6587            last_match = utf8.offset(w, sc+1+step)
6588            goto next
6589
6590          elseif crep and crep.kashida then -- Experimental
```

190

```
6591              node.set_attribute(item,
6592                Babel.attr_kashida,
6593                crep.kashida)
6594              last_match = utf8.offset(w, sc+1+step)
6595              goto next
6596
6597          elseif crep and crep.string then
6598              local str = crep.string(matches)
6599              if str == '' then  -- Gather with nil
6600                node.remove(head, item)
6601                table.remove(w_nodes, sc)
6602                w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
6603                sc = sc - 1  -- Nothing has been inserted.
6604              else
6605                local loop_first = true
6606                for s in string.utfvalues(str) do
6607                  d = node.copy(item_base)
6608                  d.char = s
6609                  if loop_first then
6610                    loop_first = false
6611                    head, new = node.insert_before(head, item, d)
6612                    if sc == 1 then
6613                      word_head = head
6614                    end
6615                    w_nodes[sc] = d
6616                    w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc+1)
6617                  else
6618                    sc = sc + 1
6619                    head, new = node.insert_before(head, item, d)
6620                    table.insert(w_nodes, sc, new)
6621                    w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc)
6622                  end
6623                  if Babel.debug then
6624                    print('.....', 'str')
6625                    Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6626                  end
6627                end  -- for
6628                node.remove(head, item)
6629              end  -- if ''
6630              last_match = utf8.offset(w, sc+1+step)
6631              goto next
6632
6633          elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
6634              d = node.new(7, 0)   -- (disc, discretionary)
6635              d.pre     = Babel.str_to_nodes(crep.pre, matches, item_base)
6636              d.post    = Babel.str_to_nodes(crep.post, matches, item_base)
6637              d.replace = Babel.str_to_nodes(crep.no, matches, item_base)
6638              d.attr = item_base.attr
6639              if crep.pre == nil then  -- TeXbook p96
6640                d.penalty = crep.penalty or tex.hyphenpenalty
6641              else
6642                d.penalty = crep.penalty or tex.exhyphenpenalty
6643              end
6644              placeholder = '|'
6645              head, new = node.insert_before(head, item, d)
6646
6647          elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
6648              -- ERROR
6649
6650          elseif crep and crep.penalty then
6651              d = node.new(14, 0)   -- (penalty, userpenalty)
6652              d.attr = item_base.attr
6653              d.penalty = crep.penalty
```

```
6654              head, new = node.insert_before(head, item, d)
6655
6656          elseif crep and crep.space then
6657            -- 655360 = 10 pt = 10 * 65536 sp
6658            d = node.new(12, 13)      -- (glue, spaceskip)
6659            local quad = font.getfont(item_base.font).size or 655360
6660            node.setglue(d, crep.space[1] * quad,
6661                            crep.space[2] * quad,
6662                            crep.space[3] * quad)
6663            if mode == 0 then
6664              placeholder = ' '
6665            end
6666            head, new = node.insert_before(head, item, d)
6667
6668          elseif crep and crep.spacefactor then
6669            d = node.new(12, 13)      -- (glue, spaceskip)
6670            local base_font = font.getfont(item_base.font)
6671            node.setglue(d,
6672              crep.spacefactor[1] * base_font.parameters['space'],
6673              crep.spacefactor[2] * base_font.parameters['space_stretch'],
6674              crep.spacefactor[3] * base_font.parameters['space_shrink'])
6675            if mode == 0 then
6676              placeholder = ' '
6677            end
6678            head, new = node.insert_before(head, item, d)
6679
6680          elseif mode == 0 and crep and crep.space then
6681            -- ERROR
6682
6683          end  -- ie replacement cases
6684
6685          -- Shared by disc, space and penalty.
6686          if sc == 1 then
6687            word_head = head
6688          end
6689          if crep.insert then
6690            w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc)
6691            table.insert(w_nodes, sc, new)
6692            last = last + 1
6693          else
6694            w_nodes[sc] = d
6695            node.remove(head, item)
6696            w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1)
6697          end
6698
6699          last_match = utf8.offset(w, sc+1+step)
6700
6701          ::next::
6702
6703        end  -- for each replacement
6704
6705        if Babel.debug then
6706            print('.....', '/')
6707            Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6708        end
6709
6710      end  -- for match
6711
6712    end  -- for patterns
6713
6714    ::next::
6715    word_head = nw
6716  end  -- for substring
```

```
6717    return head
6718 end
6719
6720 -- This table stores capture maps, numbered consecutively
6721 Babel.capture_maps = {}
6722
6723 -- The following functions belong to the next macro
6724 function Babel.capture_func(key, cap)
6725    local ret = "[[" .. cap:gsub('{([0-9])}', "]]..m[%1]..[[") .. "]]"
6726    local cnt
6727    local u = unicode.utf8
6728    ret, cnt = ret:gsub('{([0-9])|([^|]+)|(.-)}', Babel.capture_func_map)
6729    if cnt == 0 then
6730      ret = u.gsub(ret, '{(%x%x%x%x+)}',
6731            function (n)
6732              return u.char(tonumber(n, 16))
6733            end)
6734    end
6735    ret = ret:gsub("%[%[%]%]%.%.", '')
6736    ret = ret:gsub("%.%.%[%[%]%]", '')
6737    return key .. [[=function(m) return ]] .. ret .. [[ end]]
6738 end
6739
6740 function Babel.capt_map(from, mapno)
6741    return Babel.capture_maps[mapno][from] or from
6742 end
6743
6744 -- Handle the {n|abc|ABC} syntax in captures
6745 function Babel.capture_func_map(capno, from, to)
6746    local u = unicode.utf8
6747    from = u.gsub(from, '{(%x%x%x%x+)}',
6748        function (n)
6749          return u.char(tonumber(n, 16))
6750        end)
6751    to = u.gsub(to, '{(%x%x%x%x+)}',
6752        function (n)
6753          return u.char(tonumber(n, 16))
6754        end)
6755    local froms = {}
6756    for s in string.utfcharacters(from) do
6757      table.insert(froms, s)
6758    end
6759    local cnt = 1
6760    table.insert(Babel.capture_maps, {})
6761    local mlen = table.getn(Babel.capture_maps)
6762    for s in string.utfcharacters(to) do
6763      Babel.capture_maps[mlen][froms[cnt]] = s
6764      cnt = cnt + 1
6765    end
6766    return "]]..Babel.capt_map(m[" .. capno .. "]," ..
6767          (mlen) .. ").." .. "[["
6768 end
6769
6770 -- Create/Extend reversed sorted list of kashida weights:
6771 function Babel.capture_kashida(key, wt)
6772    wt = tonumber(wt)
6773    if Babel.kashida_wts then
6774      for p, q in ipairs(Babel.kashida_wts) do
6775        if wt  == q then
6776          break
6777        elseif wt > q then
6778          table.insert(Babel.kashida_wts, p, wt)
6779          break
```

```
6780        elseif table.getn(Babel.kashida_wts) == p then
6781          table.insert(Babel.kashida_wts, wt)
6782        end
6783      end
6784    else
6785      Babel.kashida_wts = { wt }
6786    end
6787    return 'kashida = ' .. wt
6788 end
6789 ⟨/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.

```
6790 ⟨∗basic-r⟩
6791 Babel = Babel or {}
6792
6793 Babel.bidi_enabled = true
6794
6795 require('babel-data-bidi.lua')
6796
6797 local characters = Babel.characters
6798 local ranges = Babel.ranges
6799
6800 local DIR = node.id("dir")
6801
6802 local function dir_mark(head, from, to, outer)
6803    dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
```

```
6804    local d = node.new(DIR)
6805    d.dir = '+' .. dir
6806    node.insert_before(head, from, d)
6807    d = node.new(DIR)
6808    d.dir = '-' .. dir
6809    node.insert_after(head, to, d)
6810 end
6811
6812 function Babel.bidi(head, ispar)
6813    local first_n, last_n            -- first and last char with nums
6814    local last_es                    -- an auxiliary 'last' used with nums
6815    local first_d, last_d            -- first and last char in L/R block
6816    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):

```
6817    local strong = ('TRT' == tex.pardir) and 'r' or 'l'
6818    local strong_lr = (strong == 'l') and 'l' or 'r'
6819    local outer = strong
6820
6821    local new_dir = false
6822    local first_dir = false
6823    local inmath = false
6824
6825    local last_lr
6826
6827    local type_n = ''
6828
6829    for item in node.traverse(head) do
6830
6831      -- three cases: glyph, dir, otherwise
6832      if item.id == node.id'glyph'
6833        or (item.id == 7 and item.subtype == 2) then
6834
6835        local itemchar
6836        if item.id == 7 and item.subtype == 2 then
6837          itemchar = item.replace.char
6838        else
6839          itemchar = item.char
6840        end
6841        local chardata = characters[itemchar]
6842        dir = chardata and chardata.d or nil
6843        if not dir then
6844          for nn, et in ipairs(ranges) do
6845            if itemchar < et[1] then
6846              break
6847            elseif itemchar <= et[2] then
6848              dir = et[3]
6849              break
6850            end
6851          end
6852        end
6853        dir = dir or 'l'
6854        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).

```
6855        if new_dir then
6856          attr_dir = 0
6857          for at in node.traverse(item.attr) do
```

```
6858            if at.number == Babel.attr_dir then
6859              attr_dir = at.value % 3
6860            end
6861          end
6862          if attr_dir == 1 then
6863            strong = 'r'
6864          elseif attr_dir == 2 then
6865            strong = 'al'
6866          else
6867            strong = 'l'
6868          end
6869          strong_lr = (strong == 'l') and 'l' or 'r'
6870          outer = strong_lr
6871          new_dir = false
6872        end
6873
6874        if dir == 'nsm' then dir = strong end              -- W1
```

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

```
6875        dir_real = dir                -- We need dir_real to set strong below
6876        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:

```
6877        if strong == 'al' then
6878          if dir == 'en' then dir = 'an' end              -- W2
6879          if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
6880          strong_lr = 'r'                                 -- W3
6881        end
```

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

```
6882      elseif item.id == node.id'dir' and not inmath then
6883        new_dir = true
6884        dir = nil
6885      elseif item.id == node.id'math' then
6886        inmath = (item.subtype == 0)
6887      else
6888        dir = nil          -- Not a char
6889      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>.

```
6890      if dir == 'en' or dir == 'an' or dir == 'et' then
6891        if dir ~= 'et' then
6892          type_n = dir
6893        end
6894        first_n = first_n or item
6895        last_n = last_es or item
6896        last_es = nil
6897      elseif dir == 'es' and last_n then -- W3+W6
6898        last_es = item
6899      elseif dir == 'cs' then             -- it's right - do nothing
6900      elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
6901        if strong_lr == 'r' and type_n ~= '' then
6902          dir_mark(head, first_n, last_n, 'r')
6903        elseif strong_lr == 'l' and first_d and type_n == 'an' then
6904          dir_mark(head, first_n, last_n, 'r')
6905          dir_mark(head, first_d, last_d, outer)
6906          first_d, last_d = nil, nil
6907        elseif strong_lr == 'l' and type_n ~= '' then
6908          last_d = last_n
```

```
6909        end
6910        type_n = ''
6911        first_n, last_n = nil, nil
6912      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:

```
6913      if dir == 'l' or dir == 'r' then
6914        if dir ~= outer then
6915          first_d = first_d or item
6916          last_d = item
6917        elseif first_d and dir ~= strong_lr then
6918          dir_mark(head, first_d, last_d, outer)
6919          first_d, last_d = nil, nil
6920        end
6921      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.

```
6922      if dir and not last_lr and dir ~= 'l' and outer == 'r' then
6923        item.char = characters[item.char] and
6924                    characters[item.char].m or item.char
6925      elseif (dir or new_dir) and last_lr ~= item then
6926        local mir = outer .. strong_lr .. (dir or outer)
6927        if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
6928          for ch in node.traverse(node.next(last_lr)) do
6929            if ch == item then break end
6930            if ch.id == node.id'glyph' and characters[ch.char] then
6931              ch.char = characters[ch.char].m or ch.char
6932            end
6933          end
6934        end
6935      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).

```
6936      if dir == 'l' or dir == 'r' then
6937        last_lr = item
6938        strong = dir_real            -- Don't search back - best save now
6939        strong_lr = (strong == 'l') and 'l' or 'r'
6940      elseif new_dir then
6941        last_lr = nil
6942      end
6943    end
```

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

```
6944    if last_lr and outer == 'r' then
6945      for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
6946        if characters[ch.char] then
6947          ch.char = characters[ch.char].m or ch.char
6948        end
6949      end
6950    end
6951    if first_n then
6952      dir_mark(head, first_n, last_n, outer)
6953    end
6954    if first_d then
6955      dir_mark(head, first_d, last_d, outer)
6956    end
```

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

```
6957    return node.prev(head) or head
6958 end
6959 ⟨/basic-r⟩
```

And here the Lua code for bidi=basic:

```
6960 ⟨∗basic⟩
6961 Babel = Babel or {}
6962
6963 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
6964
6965 Babel.fontmap = Babel.fontmap or {}
6966 Babel.fontmap[0] = {}       -- l
6967 Babel.fontmap[1] = {}       -- r
6968 Babel.fontmap[2] = {}       -- al/an
6969
6970 Babel.bidi_enabled = true
6971 Babel.mirroring_enabled = true
6972
6973 require('babel-data-bidi.lua')
6974
6975 local characters = Babel.characters
6976 local ranges = Babel.ranges
6977
6978 local DIR = node.id('dir')
6979 local GLYPH = node.id('glyph')
6980
6981 local function insert_implicit(head, state, outer)
6982    local new_state = state
6983    if state.sim and state.eim and state.sim ~= state.eim then
6984      dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
6985      local d = node.new(DIR)
6986      d.dir = '+' .. dir
6987      node.insert_before(head, state.sim, d)
6988      local d = node.new(DIR)
6989      d.dir = '-' .. dir
6990      node.insert_after(head, state.eim, d)
6991    end
6992    new_state.sim, new_state.eim = nil, nil
6993    return head, new_state
6994 end
6995
6996 local function insert_numeric(head, state)
6997    local new
6998    local new_state = state
6999    if state.san and state.ean and state.san ~= state.ean then
7000      local d = node.new(DIR)
7001      d.dir = '+TLT'
7002      _, new = node.insert_before(head, state.san, d)
7003      if state.san == state.sim then state.sim = new end
7004      local d = node.new(DIR)
7005      d.dir = '-TLT'
7006      _, new = node.insert_after(head, state.ean, d)
7007      if state.ean == state.eim then state.eim = new end
7008    end
7009    new_state.san, new_state.ean = nil, nil
7010    return head, new_state
7011 end
7012
7013 -- TODO - \hbox with an explicit dir can lead to wrong results
7014 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
7015 -- was s made to improve the situation, but the problem is the 3-dir
```

198

```
7016 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
7017 -- well.
7018
7019 function Babel.bidi(head, ispar, hdir)
7020   local d    -- d is used mainly for computations in a loop
7021   local prev_d = ''
7022   local new_d = false
7023
7024   local nodes = {}
7025   local outer_first = nil
7026   local inmath = false
7027
7028   local glue_d = nil
7029   local glue_i = nil
7030
7031   local has_en = false
7032   local first_et = nil
7033
7034   local ATDIR = Babel.attr_dir
7035
7036   local save_outer
7037   local temp = node.get_attribute(head, ATDIR)
7038   if temp then
7039     temp = temp % 3
7040     save_outer = (temp == 0 and 'l') or
7041                  (temp == 1 and 'r') or
7042                  (temp == 2 and 'al')
7043   elseif ispar then          -- Or error? Shouldn't happen
7044     save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
7045   else                       -- Or error? Shouldn't happen
7046     save_outer = ('TRT' == hdir) and 'r' or 'l'
7047   end
7048     -- when the callback is called, we are just _after_ the box,
7049     -- and the textdir is that of the surrounding text
7050 -- if not ispar and hdir ~= tex.textdir then
7051 --   save_outer = ('TRT' == hdir) and 'r' or 'l'
7052 -- end
7053   local outer = save_outer
7054   local last = outer
7055 -- 'al' is only taken into account in the first, current loop
7056   if save_outer == 'al' then save_outer = 'r' end
7057
7058   local fontmap = Babel.fontmap
7059
7060   for item in node.traverse(head) do
7061
7062     -- In what follows, #node is the last (previous) node, because the
7063     -- current one is not added until we start processing the neutrals.
7064
7065     -- three cases: glyph, dir, otherwise
7066     if item.id == GLYPH
7067       or (item.id == 7 and item.subtype == 2) then
7068
7069       local d_font = nil
7070       local item_r
7071       if item.id == 7 and item.subtype == 2 then
7072         item_r = item.replace    -- automatic discs have just 1 glyph
7073       else
7074         item_r = item
7075       end
7076       local chardata = characters[item_r.char]
7077       d = chardata and chardata.d or nil
7078       if not d or d == 'nsm' then
```

```
7079            for nn, et in ipairs(ranges) do
7080              if item_r.char < et[1] then
7081                break
7082              elseif item_r.char <= et[2] then
7083                if not d then d = et[3]
7084                elseif d == 'nsm' then d_font = et[3]
7085                end
7086                break
7087              end
7088            end
7089          end
7090          d = d or 'l'
7091
7092          -- A short 'pause' in bidi for mapfont
7093          d_font = d_font or d
7094          d_font = (d_font == 'l' and 0) or
7095                   (d_font == 'nsm' and 0) or
7096                   (d_font == 'r' and 1) or
7097                   (d_font == 'al' and 2) or
7098                   (d_font == 'an' and 2) or nil
7099        if d_font and fontmap and fontmap[d_font][item_r.font] then
7100          item_r.font = fontmap[d_font][item_r.font]
7101        end
7102
7103        if new_d then
7104          table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
7105          if inmath then
7106            attr_d = 0
7107          else
7108            attr_d = node.get_attribute(item, ATDIR)
7109            attr_d = attr_d % 3
7110          end
7111          if attr_d == 1 then
7112            outer_first = 'r'
7113            last = 'r'
7114          elseif attr_d == 2 then
7115            outer_first = 'r'
7116            last = 'al'
7117          else
7118            outer_first = 'l'
7119            last = 'l'
7120          end
7121          outer = last
7122          has_en = false
7123          first_et = nil
7124          new_d = false
7125        end
7126
7127        if glue_d then
7128          if (d == 'l' and 'l' or 'r') ~= glue_d then
7129            table.insert(nodes, {glue_i, 'on', nil})
7130          end
7131          glue_d = nil
7132          glue_i = nil
7133        end
7134
7135      elseif item.id == DIR then
7136        d = nil
7137        if head ~= item then new_d = true end
7138
7139      elseif item.id == node.id'glue' and item.subtype == 13 then
7140        glue_d = d
7141        glue_i = item
```

```
7142        d = nil
7143
7144      elseif item.id == node.id'math' then
7145        inmath = (item.subtype == 0)
7146
7147      else
7148        d = nil
7149      end
7150
7151      -- AL <= EN/ET/ES      -- W2 + W3 + W6
7152      if last == 'al' and d == 'en' then
7153        d = 'an'            -- W3
7154      elseif last == 'al' and (d == 'et' or d == 'es') then
7155        d = 'on'            -- W6
7156      end
7157
7158      -- EN + CS/ES + EN      -- W4
7159      if d == 'en' and #nodes >= 2 then
7160        if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
7161            and nodes[#nodes-1][2] == 'en' then
7162          nodes[#nodes][2] = 'en'
7163        end
7164      end
7165
7166      -- AN + CS + AN         -- W4 too, because uax9 mixes both cases
7167      if d == 'an' and #nodes >= 2 then
7168        if (nodes[#nodes][2] == 'cs')
7169            and nodes[#nodes-1][2] == 'an' then
7170          nodes[#nodes][2] = 'an'
7171        end
7172      end
7173
7174      -- ET/EN                -- W5 + W7->l / W6->on
7175      if d == 'et' then
7176        first_et = first_et or (#nodes + 1)
7177      elseif d == 'en' then
7178        has_en = true
7179        first_et = first_et or (#nodes + 1)
7180      elseif first_et then        -- d may be nil here !
7181        if has_en then
7182          if last == 'l' then
7183            temp = 'l'      -- W7
7184          else
7185            temp = 'en'     -- W5
7186          end
7187        else
7188          temp = 'on'       -- W6
7189        end
7190        for e = first_et, #nodes do
7191          if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
7192        end
7193        first_et = nil
7194        has_en = false
7195      end
7196
7197      -- Force mathdir in math if ON (currently works as expected only
7198      -- with 'l')
7199      if inmath and d == 'on' then
7200        d = ('TRT' == tex.mathdir) and 'r' or 'l'
7201      end
7202
7203      if d then
7204        if d == 'al' then
```

```
7205        d = 'r'
7206        last = 'al'
7207      elseif d == 'l' or d == 'r' then
7208        last = d
7209      end
7210      prev_d = d
7211      table.insert(nodes, {item, d, outer_first})
7212    end
7213
7214    outer_first = nil
7215
7216  end
7217
7218  -- TODO -- repeated here in case EN/ET is the last node. Find a
7219  -- better way of doing things:
7220  if first_et then        -- dir may be nil here !
7221    if has_en then
7222      if last == 'l' then
7223        temp = 'l'     -- W7
7224      else
7225        temp = 'en'    -- W5
7226      end
7227    else
7228      temp = 'on'      -- W6
7229    end
7230    for e = first_et, #nodes do
7231      if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
7232    end
7233  end
7234
7235  -- dummy node, to close things
7236  table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
7237
7238  --------------  NEUTRAL -----------------
7239
7240  outer = save_outer
7241  last = outer
7242
7243  local first_on = nil
7244
7245  for q = 1, #nodes do
7246    local item
7247
7248    local outer_first = nodes[q][3]
7249    outer = outer_first or outer
7250    last = outer_first or last
7251
7252    local d = nodes[q][2]
7253    if d == 'an' or d == 'en' then d = 'r' end
7254    if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
7255
7256    if d == 'on' then
7257      first_on = first_on or q
7258    elseif first_on then
7259      if last == d then
7260        temp = d
7261      else
7262        temp = outer
7263      end
7264      for r = first_on, q - 1 do
7265        nodes[r][2] = temp
7266        item = nodes[r][1]     -- MIRRORING
7267        if Babel.mirroring_enabled and item.id == GLYPH
```

202

```
7268              and temp == 'r' and characters[item.char] then
7269          local font_mode = ''
7270          if font.fonts[item.font].properties then
7271            font_mode = font.fonts[item.font].properties.mode
7272          end
7273          if font_mode ~= 'harf' and font_mode ~= 'plug' then
7274            item.char = characters[item.char].m or item.char
7275          end
7276        end
7277      end
7278      first_on = nil
7279    end
7280
7281    if d == 'r' or d == 'l' then last = d end
7282  end
7283
7284  -------------  IMPLICIT, REORDER ----------------
7285
7286  outer = save_outer
7287  last = outer
7288
7289  local state = {}
7290  state.has_r = false
7291
7292  for q = 1, #nodes do
7293
7294    local item = nodes[q][1]
7295
7296    outer = nodes[q][3] or outer
7297
7298    local d = nodes[q][2]
7299
7300    if d == 'nsm' then d = last end               -- W1
7301    if d == 'en' then d = 'an' end
7302    local isdir = (d == 'r' or d == 'l')
7303
7304    if outer == 'l' and d == 'an' then
7305      state.san = state.san or item
7306      state.ean = item
7307    elseif state.san then
7308      head, state = insert_numeric(head, state)
7309    end
7310
7311    if outer == 'l' then
7312      if d == 'an' or d == 'r' then      -- im -> implicit
7313        if d == 'r' then state.has_r = true end
7314        state.sim = state.sim or item
7315        state.eim = item
7316      elseif d == 'l' and state.sim and state.has_r then
7317        head, state = insert_implicit(head, state, outer)
7318      elseif d == 'l' then
7319        state.sim, state.eim, state.has_r = nil, nil, false
7320      end
7321    else
7322      if d == 'an' or d == 'l' then
7323        if nodes[q][3] then -- nil except after an explicit dir
7324          state.sim = item  -- so we move sim 'inside' the group
7325        else
7326          state.sim = state.sim or item
7327        end
7328        state.eim = item
7329      elseif d == 'r' and state.sim then
7330        head, state = insert_implicit(head, state, outer)
```

```
7331        elseif d == 'r' then
7332          state.sim, state.eim = nil, nil
7333        end
7334      end
7335
7336      if isdir then
7337        last = d            -- Don't search back - best save now
7338      elseif d == 'on' and state.san  then
7339        state.san = state.san or item
7340        state.ean = item
7341      end
7342
7343   end
7344
7345   return node.prev(head) or head
7346 end
7347 ⟨/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.

```
7348 ⟨∗nil⟩
7349 \ProvidesLanguage{nil}[⟨⟨date⟩⟩ ⟨⟨version⟩⟩ Nil language]
7350 \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.

```
7351 \ifx\l@nil\@undefined
7352   \newlanguage\l@nil
7353   \@namedef{bbl@hyphendata@\the\l@nil}{{}{}}% Remove warning
7354   \let\bbl@elt\relax
7355   \edef\bbl@languages{%  Add it to the list of languages
7356     \bbl@languages\bbl@elt{nil}{\the\l@nil}{}{}}
7357 \fi
```

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

```
7358 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}
```

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

\captionnil
\datenil
```
7359 \let\captionsnil\@empty
7360 \let\datenil\@empty
```

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

```
7361 \def\bbl@inidata@nil{%
7362   \bbl@elt{identification}{tag.ini}{und}%
7363   \bbl@elt{identification}{load.level}{0}%
7364   \bbl@elt{identification}{charset}{utf8}%
7365   \bbl@elt{identification}{version}{1.0}%
7366   \bbl@elt{identification}{date}{2022-05-16}%
7367   \bbl@elt{identification}{name.local}{nil}%
7368   \bbl@elt{identification}{name.english}{nil}%
7369   \bbl@elt{identification}{name.babel}{nil}%
7370   \bbl@elt{identification}{tag.bcp47}{und}%
7371   \bbl@elt{identification}{language.tag.bcp47}{und}%
7372   \bbl@elt{identification}{tag.opentype}{dflt}%
7373   \bbl@elt{identification}{script.name}{Latin}%
7374   \bbl@elt{identification}{script.tag.bcp47}{Latn}%
7375   \bbl@elt{identification}{script.tag.opentype}{DFLT}%
7376   \bbl@elt{identification}{level}{1}%
7377   \bbl@elt{identification}{encodings}{}%
7378   \bbl@elt{identification}{derivate}{no}}
7379 \@namedef{bbl@tbcp@nil}{und}
7380 \@namedef{bbl@lbcp@nil}{und}
7381 \@namedef{bbl@lotf@nil}{dflt}
7382 \@namedef{bbl@elname@nil}{nil}
7383 \@namedef{bbl@lname@nil}{nil}
7384 \@namedef{bbl@esname@nil}{Latin}
7385 \@namedef{bbl@sname@nil}{Latin}
7386 \@namedef{bbl@sbcp@nil}{Latn}
7387 \@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.

```
7388 \ldf@finish{nil}
7389 ⟨/nil⟩
```

15   Calendars

The code for specific calendars are placed in the specific files, loaded when requested by an ini file in the identification section with require.calendars.
Start with function to compute the Julian day. It's based on the little library calendar.js, by John Walker, in the public domain.

```
7390 ⟨⟨*Compute Julian day⟩⟩ ≡
7391 \def\bbl@fpmod#1#2{(#1-#2*floor(#1/#2))}
7392 \def\bbl@cs@gregleap#1{%
7393   (\bbl@fpmod{#1}{4} == 0) &&
7394     (!((\bbl@fpmod{#1}{100} == 0) && (\bbl@fpmod{#1}{400} != 0)))}
7395 \def\bbl@cs@jd#1#2#3{% year, month, day
7396   \fp_eval:n{ 1721424.5  + (365 * (#1 - 1)) +
7397     floor((#1 - 1) / 4)   + (-floor((#1 - 1) / 100)) +
7398     floor((#1 - 1) / 400) + floor((((367 * #2) - 362) / 12) +
7399     ((#2 <= 2) ? 0 : (\bbl@cs@gregleap{#1} ? -1 : -2)) + #3) }}
7400 ⟨⟨/Compute Julian day⟩⟩
```

15.1   Islamic

The code for the Civil calendar is based on it, too.

```
7401 ⟨*ca-islamic⟩
7402 \ExplSyntaxOn
7403 ⟨⟨Compute Julian day⟩⟩
7404 % == islamic (default)
7405 % Not yet implemented
7406 \def\bbl@ca@islamic#1-#2-#3\@@#4#5#6{}
```

The Civil calendar.

```
7407 \def\bbl@cs@isltojd#1#2#3{ % year, month, day
7408   ((#3 + ceil(29.5 * (#2 - 1)) +
7409   (#1 - 1) * 354 + floor((3 + (11 * #1)) / 30) +
7410   1948439.5) - 1) }
7411 \@namedef{bbl@ca@islamic-civil++}{\bbl@ca@islamicvl@x{+2}}
7412 \@namedef{bbl@ca@islamic-civil+}{\bbl@ca@islamicvl@x{+1}}
7413 \@namedef{bbl@ca@islamic-civil}{\bbl@ca@islamicvl@x{}}
7414 \@namedef{bbl@ca@islamic-civil-}{\bbl@ca@islamicvl@x{-1}}
7415 \@namedef{bbl@ca@islamic-civil--}{\bbl@ca@islamicvl@x{-2}}
7416 \def\bbl@ca@islamicvl@x#1#2-#3-#4\@@#5#6#7{%
7417   \edef\bbl@tempa{%
7418     \fp_eval:n{ floor(\bbl@cs@jd{#2}{#3}{#4})+0.5 #1}}%
7419   \edef#5{%
7420     \fp_eval:n{ floor(((30*(\bbl@tempa-1948439.5)) + 10646)/10631) }}%
7421   \edef#6{\fp_eval:n{
7422     min(12,ceil((\bbl@tempa-(29+\bbl@cs@isltojd{#5}{1}{1}))/29.5)+1) }}%
7423   \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).

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

```
7464        \bbl@cs@jd{#2}{#3}{#4} + 0.5 - 2400000 #1}}%
7465    \count@\@ne
7466    \bbl@foreach\bbl@cs@umalqura@data{%
7467        \advance\count@\@ne
7468        \ifnum##1>\bbl@tempd\else
7469            \edef\bbl@tempe{\the\count@}%
7470            \edef\bbl@tempb{##1}%
7471        \fi}%
7472    \edef\bbl@templ{\fp_eval:n{ \bbl@tempe + 16260 + 949 }}% month~lunar
7473    \edef\bbl@tempa{\fp_eval:n{ floor((\bbl@templ - 1 ) / 12) }}% annus
7474    \edef#5{\fp_eval:n{ \bbl@tempa + 1   }}%
7475    \edef#6{\fp_eval:n{ \bbl@templ - (12 * \bbl@tempa) }}%
7476    \edef#7{\fp_eval:n{ \bbl@tempd - \bbl@tempb + 1 }}}
7477 \ExplSyntaxOff
7478 \bbl@add\bbl@precalendar{%
7479    \bbl@replace\bbl@ld@calendar{-civil}{}%
7480    \bbl@replace\bbl@ld@calendar{-umalqura}{}%
7481    \bbl@replace\bbl@ld@calendar{+}{}%
7482    \bbl@replace\bbl@ld@calendar{-}{}}
7483 ⟨/ca-islamic⟩
```

16   Hebrew

This is basically the set of macros written by Michail Rozman in 1991, with corrections and adaptions by Rama Porrat, Misha, Dan Haran and Boris Lavva. This must be eventually replaced by computations with l3fp. An explanation of what's going on can be found in hebcal.sty

```
7484 ⟨*ca-hebrew⟩
7485 \newcount\bbl@cntcommon
7486 \def\bbl@remainder#1#2#3{%
7487    #3=#1\relax
7488    \divide #3 by #2\relax
7489    \multiply #3 by -#2\relax
7490    \advance #3 by #1\relax}%
7491 \newif\ifbbl@divisible
7492 \def\bbl@checkifdivisible#1#2{%
7493    {\countdef\tmp=0
7494    \bbl@remainder{#1}{#2}{\tmp}%
7495    \ifnum \tmp=0
7496        \global\bbl@divisibletrue
7497    \else
7498        \global\bbl@divisiblefalse
7499    \fi}}
7500 \newif\ifbbl@gregleap
7501 \def\bbl@ifgregleap#1{%
7502    \bbl@checkifdivisible{#1}{4}%
7503    \ifbbl@divisible
7504        \bbl@checkifdivisible{#1}{100}%
7505        \ifbbl@divisible
7506            \bbl@checkifdivisible{#1}{400}%
7507            \ifbbl@divisible
7508                \bbl@gregleaptrue
7509            \else
7510                \bbl@gregleapfalse
7511            \fi
7512        \else
7513            \bbl@gregleaptrue
7514        \fi
7515    \else
7516        \bbl@gregleapfalse
7517    \fi
7518    \ifbbl@gregleap}
7519 \def\bbl@gregdayspriormonths#1#2#3{%
```

207

```
7520     {#3=\ifcase #1 0 \or 0 \or 31 \or 59 \or 90 \or 120 \or 151 \or
7521          181 \or 212 \or 243 \or 273 \or 304 \or 334 \fi
7522       \bbl@ifgregleap{#2}%
7523          \ifnum #1 > 2
7524             \advance #3 by 1
7525          \fi
7526       \fi
7527       \global\bbl@cntcommon=#3}%
7528     #3=\bbl@cntcommon}
7529 \def\bbl@gregdaysprioryears#1#2{%
7530   {\countdef\tmpc=4
7531    \countdef\tmpb=2
7532    \tmpb=#1\relax
7533    \advance \tmpb by -1
7534    \tmpc=\tmpb
7535    \multiply \tmpc by 365
7536    #2=\tmpc
7537    \tmpc=\tmpb
7538    \divide \tmpc by 4
7539    \advance #2 by \tmpc
7540    \tmpc=\tmpb
7541    \divide \tmpc by 100
7542    \advance #2 by -\tmpc
7543    \tmpc=\tmpb
7544    \divide \tmpc by 400
7545    \advance #2 by \tmpc
7546    \global\bbl@cntcommon=#2\relax}%
7547   #2=\bbl@cntcommon}
7548 \def\bbl@absfromgreg#1#2#3#4{%
7549   {\countdef\tmpd=0
7550    #4=#1\relax
7551    \bbl@gregdayspriormonths{#2}{#3}{\tmpd}%
7552    \advance #4 by \tmpd
7553    \bbl@gregdaysprioryears{#3}{\tmpd}%
7554    \advance #4 by \tmpd
7555    \global\bbl@cntcommon=#4\relax}%
7556   #4=\bbl@cntcommon}
7557 \newif\ifbbl@hebrleap
7558 \def\bbl@checkleaphebryear#1{%
7559   {\countdef\tmpa=0
7560    \countdef\tmpb=1
7561    \tmpa=#1\relax
7562    \multiply \tmpa by 7
7563    \advance \tmpa by 1
7564    \bbl@remainder{\tmpa}{19}{\tmpb}%
7565    \ifnum \tmpb < 7
7566       \global\bbl@hebrleaptrue
7567    \else
7568       \global\bbl@hebrleapfalse
7569    \fi}}
7570 \def\bbl@hebrelapsedmonths#1#2{%
7571   {\countdef\tmpa=0
7572    \countdef\tmpb=1
7573    \countdef\tmpc=2
7574    \tmpa=#1\relax
7575    \advance \tmpa by -1
7576    #2=\tmpa
7577    \divide #2 by 19
7578    \multiply #2 by 235
7579    \bbl@remainder{\tmpa}{19}{\tmpb}% \tmpa=years%19-years this cycle
7580    \tmpc=\tmpb
7581    \multiply \tmpb by 12
7582    \advance #2 by \tmpb
```

```
7583    \multiply \tmpc by 7
7584    \advance \tmpc by 1
7585    \divide \tmpc by 19
7586    \advance #2 by \tmpc
7587    \global\bbl@cntcommon=#2}%
7588   #2=\bbl@cntcommon}
7589 \def\bbl@hebrelapseddays#1#2{%
7590   {\countdef\tmpa=0
7591    \countdef\tmpb=1
7592    \countdef\tmpc=2
7593    \bbl@hebrelapsedmonths{#1}{#2}%
7594    \tmpa=#2\relax
7595    \multiply \tmpa by 13753
7596    \advance \tmpa by 5604
7597    \bbl@remainder{\tmpa}{25920}{\tmpc}% \tmpc == ConjunctionParts
7598    \divide \tmpa by 25920
7599    \multiply #2 by 29
7600    \advance #2 by 1
7601    \advance #2 by \tmpa
7602    \bbl@remainder{#2}{7}{\tmpa}%
7603    \ifnum \tmpc < 19440
7604        \ifnum \tmpc < 9924
7605        \else
7606            \ifnum \tmpa=2
7607                \bbl@checkleaphebryear{#1}% of a common year
7608                \ifbbl@hebrleap
7609                \else
7610                    \advance #2 by 1
7611                \fi
7612            \fi
7613        \fi
7614        \ifnum \tmpc < 16789
7615        \else
7616            \ifnum \tmpa=1
7617                \advance #1 by -1
7618                \bbl@checkleaphebryear{#1}% at the end of leap year
7619                \ifbbl@hebrleap
7620                    \advance #2 by 1
7621                \fi
7622            \fi
7623        \fi
7624    \else
7625        \advance #2 by 1
7626    \fi
7627    \bbl@remainder{#2}{7}{\tmpa}%
7628    \ifnum \tmpa=0
7629        \advance #2 by 1
7630    \else
7631        \ifnum \tmpa=3
7632            \advance #2 by 1
7633        \else
7634            \ifnum \tmpa=5
7635                \advance #2 by 1
7636            \fi
7637        \fi
7638    \fi
7639    \global\bbl@cntcommon=#2\relax}%
7640   #2=\bbl@cntcommon}
7641 \def\bbl@daysinhebryear#1#2{%
7642   {\countdef\tmpe=12
7643    \bbl@hebrelapseddays{#1}{\tmpe}%
7644    \advance #1 by 1
7645    \bbl@hebrelapseddays{#1}{#2}%
```

```
7646    \advance #2 by -\tmpe
7647    \global\bbl@cntcommon=#2}%
7648   #2=\bbl@cntcommon}
7649 \def\bbl@hebrdayspriormonths#1#2#3{%
7650   {\countdef\tmpf= 14
7651    #3=\ifcase #1\relax
7652           0 \or
7653           0 \or
7654          30 \or
7655          59 \or
7656          89 \or
7657         118 \or
7658         148 \or
7659         148 \or
7660         177 \or
7661         207 \or
7662         236 \or
7663         266 \or
7664         295 \or
7665         325 \or
7666         400
7667    \fi
7668    \bbl@checkleaphebryear{#2}%
7669    \ifbbl@hebrleap
7670        \ifnum #1 > 6
7671            \advance #3 by 30
7672        \fi
7673    \fi
7674    \bbl@daysinhebryear{#2}{\tmpf}%
7675    \ifnum #1 > 3
7676        \ifnum \tmpf=353
7677            \advance #3 by -1
7678        \fi
7679        \ifnum \tmpf=383
7680            \advance #3 by -1
7681        \fi
7682    \fi
7683    \ifnum #1 > 2
7684        \ifnum \tmpf=355
7685            \advance #3 by 1
7686        \fi
7687        \ifnum \tmpf=385
7688            \advance #3 by 1
7689        \fi
7690    \fi
7691    \global\bbl@cntcommon=#3\relax}%
7692   #3=\bbl@cntcommon}
7693 \def\bbl@absfromhebr#1#2#3#4{%
7694   {#4=#1\relax
7695    \bbl@hebrdayspriormonths{#2}{#3}{#1}%
7696    \advance #4 by #1\relax
7697    \bbl@hebrelapseddays{#3}{#1}%
7698    \advance #4 by #1\relax
7699    \advance #4 by -1373429
7700    \global\bbl@cntcommon=#4\relax}%
7701   #4=\bbl@cntcommon}
7702 \def\bbl@hebrfromgreg#1#2#3#4#5#6{%
7703   {\countdef\tmpx= 17
7704    \countdef\tmpy= 18
7705    \countdef\tmpz= 19
7706    #6=#3\relax
7707    \global\advance #6 by 3761
7708    \bbl@absfromgreg{#1}{#2}{#3}{#4}%
```

```
7709    \tmpz=1  \tmpy=1
7710    \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
7711    \ifnum \tmpx > #4\relax
7712        \global\advance #6 by -1
7713        \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
7714    \fi
7715    \advance #4 by -\tmpx
7716    \advance #4 by 1
7717    #5=#4\relax
7718    \divide #5 by 30
7719    \loop
7720        \bbl@hebrdayspriormonths{#5}{#6}{\tmpx}%
7721        \ifnum \tmpx < #4\relax
7722            \advance #5 by 1
7723            \tmpy=\tmpx
7724    \repeat
7725    \global\advance #5 by -1
7726    \global\advance #4 by -\tmpy}}
7727 \newcount\bbl@hebrday \newcount\bbl@hebrmonth \newcount\bbl@hebryear
7728 \newcount\bbl@gregday \newcount\bbl@gregmonth \newcount\bbl@gregyear
7729 \def\bbl@ca@hebrew#1-#2-#3\@@#4#5#6{%
7730    \bbl@gregday=#3\relax \bbl@gregmonth=#2\relax \bbl@gregyear=#1\relax
7731    \bbl@hebrfromgreg
7732        {\bbl@gregday}{\bbl@gregmonth}{\bbl@gregyear}%
7733        {\bbl@hebrday}{\bbl@hebrmonth}{\bbl@hebryear}%
7734    \edef#4{\the\bbl@hebryear}%
7735    \edef#5{\the\bbl@hebrmonth}%
7736    \edef#6{\the\bbl@hebrday}}
7737 ⟨/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).

```
7738 ⟨*ca-persian⟩
7739 \ExplSyntaxOn
7740 ⟨⟨Compute Julian day⟩⟩
7741 \def\bbl@cs@firstjal@xx{2012,2016,2020,2024,2028,2029,% March 20
7742    2032,2033,2036,2037,2040,2041,2044,2045,2048,2049}
7743 \def\bbl@ca@persian#1-#2-#3\@@#4#5#6{%
7744    \edef\bbl@tempa{#1}%  20XX-03-\bbl@tempe = 1 farvardin:
7745    \ifnum\bbl@tempa>2012 \ifnum\bbl@tempa<2051
7746        \bbl@afterfi\expandafter\@gobble
7747    \fi\fi
7748        {\bbl@error{Year~out~of~range}{The~allowed~range~is~2013-2050}}%
7749    \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
7750    \ifin@\def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
7751    \edef\bbl@tempc{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{#2}{#3}+.5}}% current
7752    \edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}% begin
7753    \ifnum\bbl@tempc<\bbl@tempb
7754        \edef\bbl@tempa{\fp_eval:n{\bbl@tempa-1}}% go back 1 year and redo
7755        \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
7756        \ifin@\def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
7757        \edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}%
7758    \fi
7759    \edef#4{\fp_eval:n{\bbl@tempa-621}}% set Jalali year
7760    \edef#6{\fp_eval:n{\bbl@tempc-\bbl@tempb+1}}% days from 1 farvardin
7761    \edef#5{\fp_eval:n{% set Jalali month
7762        (#6 <= 186) ? ceil(#6 / 31) : ceil((#6 - 6) / 30)}}
```

```
7763   \edef#6{\fp_eval:n{% set Jalali day
7764     (#6 - ((#5 <= 7) ? ((#5 - 1) * 31) : (((#5 - 1) * 30) + 6)))}}}
7765 \ExplSyntaxOff
7766 ⟨/ca-persian⟩
```

18   Coptic and Ethiopic

Adapted from `jquery.calendars.package-1.1.4`, written by Keith Wood, 2010. Dual license: GPL and MIT. The only difference is the epoch.

```
7767 ⟨∗ca-coptic⟩
7768 \ExplSyntaxOn
7769 ⟨⟨Compute Julian day⟩⟩
7770 \def\bbl@ca@coptic#1-#2-#3\@@#4#5#6{%
7771   \edef\bbl@tempd{\fp_eval:n{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
7772   \edef\bbl@tempc{\fp_eval:n{\bbl@tempd - 1825029.5}}%
7773   \edef#4{\fp_eval:n{%
7774     floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
7775   \edef\bbl@tempc{\fp_eval:n{%
7776      \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1825029.5}}%
7777   \edef#5{\fp_eval:n{floor(\bbl@tempc / 30) + 1}}%
7778   \edef#6{\fp_eval:n{\bbl@tempc - (#5 - 1) * 30 + 1}}}
7779 \ExplSyntaxOff
7780 ⟨/ca-coptic⟩
7781 ⟨∗ca-ethiopic⟩
7782 \ExplSyntaxOn
7783 ⟨⟨Compute Julian day⟩⟩
7784 \def\bbl@ca@ethiopic#1-#2-#3\@@#4#5#6{%
7785   \edef\bbl@tempd{\fp_eval:n{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
7786   \edef\bbl@tempc{\fp_eval:n{\bbl@tempd - 1724220.5}}%
7787   \edef#4{\fp_eval:n{%
7788     floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
7789   \edef\bbl@tempc{\fp_eval:n{%
7790      \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1724220.5}}%
7791   \edef#5{\fp_eval:n{floor(\bbl@tempc / 30) + 1}}%
7792   \edef#6{\fp_eval:n{\bbl@tempc - (#5 - 1) * 30 + 1}}}
7793 \ExplSyntaxOff
7794 ⟨/ca-ethiopic⟩
```

19   Buddhist

That's very simple.

```
7795 ⟨∗ca-buddhist⟩
7796 \def\bbl@ca@buddhist#1-#2-#3\@@#4#5#6{%
7797   \edef#4{\number\numexpr#1+543\relax}%
7798   \edef#5{#2}%
7799   \edef#6{#3}}
7800 ⟨/ca-buddhist⟩
```

20   Support for Plain TeX (`plain.def`)

20.1   Not renaming `hyphen.tex`

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

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

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

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

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

```
7801 ⟨∗bplain | blplain⟩
7802 \catcode`\{=1 % left brace is begin-group character
7803 \catcode`\}=2 % right brace is end-group character
7804 \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).

```
7805 \openin 0 hyphen.cfg
7806 \ifeof0
7807 \else
7808   \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.

```
7809   \def\input #1 {%
7810     \let\input\a
7811     \a hyphen.cfg
7812     \let\a\undefined
7813   }
7814 \fi
7815 ⟨/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.

```
7816 ⟨bplain⟩\a plain.tex
7817 ⟨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.

```
7818 ⟨bplain⟩\def\fmtname{babel-plain}
7819 ⟨blplain⟩\def\fmtname{babel-lplain}
```

When you are using a different format, based on plain.tex you can make a copy of blplain.tex, rename it and replace `plain.tex` with the name of your format file.

20.2   Emulating some LaTeX features

The file `babel.def` expects some definitions made in the LaTeX 2_ε style file. So, in Plain we must provide at least some predefined values as well some tools to set them (even if not all options are available). There are no package options, and therefore and alternative mechanism is provided. For the moment, only `\babeloptionstrings` and `\babeloptionmath` are provided, which can be defined before loading babel. `\BabelModifiers` can be set too (but not sure it works).

```
7820 ⟨⟨∗Emulate LaTeX⟩⟩ ≡
7821 \def\@empty{}
7822 \def\loadlocalcfg#1{%
7823   \openin0#1.cfg
7824   \ifeof0
7825     \closein0
7826   \else
7827     \closein0
7828     {\immediate\write16{*************************************}%
7829      \immediate\write16{* Local config file #1.cfg used}%
7830      \immediate\write16{*}%
7831     }
7832     \input #1.cfg\relax
7833   \fi
7834   \@endofldf}
```

20.3 General tools

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

```
7835 \long\def\@firstofone#1{#1}
7836 \long\def\@firstoftwo#1#2{#1}
7837 \long\def\@secondoftwo#1#2{#2}
7838 \def\@nnil{\@nil}
7839 \def\@gobbletwo#1#2{}
7840 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
7841 \def\@star@or@long#1{%
7842   \@ifstar
7843   {\let\l@ngrel@x\relax#1}%
7844   {\let\l@ngrel@x\long#1}}
7845 \let\l@ngrel@x\relax
7846 \def\@car#1#2\@nil{#1}
7847 \def\@cdr#1#2\@nil{#2}
7848 \let\@typeset@protect\relax
7849 \let\protected@edef\edef
7850 \long\def\@gobble#1{}
7851 \edef\@backslashchar{\expandafter\@gobble\string\\}
7852 \def\strip@prefix#1>{}
7853 \def\g@addto@macro#1#2{{%
7854     \toks@\expandafter{#1#2}%
7855     \xdef#1{\the\toks@}}}
7856 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
7857 \def\@nameuse#1{\csname #1\endcsname}
7858 \def\@ifundefined#1{%
7859   \expandafter\ifx\csname#1\endcsname\relax
7860     \expandafter\@firstoftwo
7861   \else
7862     \expandafter\@secondoftwo
7863   \fi}
7864 \def\@expandtwoargs#1#2#3{%
7865   \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
7866 \def\zap@space#1 #2{%
7867   #1%
7868   \ifx#2\@empty\else\expandafter\zap@space\fi
7869   #2}
7870 \let\bbl@trace\@gobble
7871 \def\bbl@error#1#2{%
7872   \begingroup
7873     \newlinechar=`\^^J
7874     \def\\{^^J(babel) }%
7875     \errhelp{#2}\errmessage{\\#1}%
7876   \endgroup}
7877 \def\bbl@warning#1{%
7878   \begingroup
7879     \newlinechar=`\^^J
7880     \def\\{^^J(babel) }%
7881     \message{\\#1}%
7882   \endgroup}
7883 \let\bbl@infowarn\bbl@warning
7884 \def\bbl@info#1{%
7885   \begingroup
7886     \newlinechar=`\^^J
7887     \def\\{^^J}%
7888     \wlog{#1}%
7889   \endgroup}
```

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

```
7890 \ifx\@preamblecmds\@undefined
7891   \def\@preamblecmds{}
```

```
7892 \fi
7893 \def\@onlypreamble#1{%
7894    \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
7895       \@preamblecmds\do#1}}
7896 \@onlypreamble\@onlypreamble
```

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

```
7897 \def\begindocument{%
7898    \@begindocumenthook
7899    \global\let\@begindocumenthook\@undefined
7900    \def\do##1{\global\let##1\@undefined}%
7901    \@preamblecmds
7902    \global\let\do\noexpand}
7903 \ifx\@begindocumenthook\@undefined
7904    \def\@begindocumenthook{}
7905 \fi
7906 \@onlypreamble\@begindocumenthook
7907 \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.

```
7908 \def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
7909 \@onlypreamble\AtEndOfPackage
7910 \def\@endofldf{}
7911 \@onlypreamble\@endofldf
7912 \let\bbl@afterlang\@empty
7913 \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.

```
7914 \catcode`\&=\z@
7915 \ifx&if@filesw\@undefined
7916    \expandafter\let\csname if@filesw\expandafter\endcsname
7917       \csname iffalse\endcsname
7918 \fi
7919 \catcode`\&=4
```

Mimick LATEX's commands to define control sequences.

```
7920 \def\newcommand{\@star@or@long\new@command}
7921 \def\new@command#1{%
7922    \@testopt{\@newcommand#1}0}
7923 \def\@newcommand#1[#2]{%
7924    \@ifnextchar [{\@xargdef#1[#2]}%
7925                 {\@argdef#1[#2]}}
7926 \long\def\@argdef#1[#2]#3{%
7927    \@yargdef#1\@ne{#2}{#3}}
7928 \long\def\@xargdef#1[#2][#3]#4{%
7929    \expandafter\def\expandafter#1\expandafter{%
7930       \expandafter\@protected@testopt\expandafter #1%
7931       \csname\string#1\expandafter\endcsname{#3}}%
7932    \expandafter\@yargdef \csname\string#1\endcsname
7933    \tw@{#2}{#4}}
7934 \long\def\@yargdef#1#2#3{%
7935    \@tempcnta#3\relax
7936    \advance \@tempcnta \@ne
7937    \let\@hash@\relax
7938    \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
7939    \@tempcntb #2%
7940    \@whilenum\@tempcntb <\@tempcnta
7941    \do{%
7942       \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
7943       \advance\@tempcntb \@ne}%
```

215

```
7944     \let\@hash@##%
7945     \l@ngrel@x\expandafter\def\expandafter#1\reserved@a}
7946 \def\providecommand{\@star@or@long\provide@command}
7947 \def\provide@command#1{%
7948     \begingroup
7949       \escapechar\m@ne\xdef\@gtempa{{\string#1}}%
7950     \endgroup
7951     \expandafter\@ifundefined\@gtempa
7952       {\def\reserved@a{\new@command#1}}%
7953       {\let\reserved@a\relax
7954        \def\reserved@a{\new@command\reserved@a}}%
7955     \reserved@a}%

7956 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
7957 \def\declare@robustcommand#1{%
7958     \edef\reserved@a{\string#1}%
7959     \def\reserved@b{#1}%
7960     \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
7961     \edef#1{%
7962       \ifx\reserved@a\reserved@b
7963         \noexpand\x@protect
7964         \noexpand#1%
7965       \fi
7966       \noexpand\protect
7967       \expandafter\noexpand\csname
7968         \expandafter\@gobble\string#1 \endcsname
7969     }%
7970     \expandafter\new@command\csname
7971       \expandafter\@gobble\string#1 \endcsname
7972 }
7973 \def\x@protect#1{%
7974     \ifx\protect\@typeset@protect\else
7975       \@x@protect#1%
7976     \fi
7977 }
7978 \catcode`\&=\z@  % Trick to hide conditionals
7979   \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`.

```
7980   \def\bbl@tempa{\csname newif\endcsname&ifin@}
7981 \catcode`\&=4
7982 \ifx\in@\@undefined
7983   \def\in@#1#2{%
7984     \def\in@@##1#1##2##3\in@@{%
7985       \ifx\in@##2\in@false\else\in@true\fi}%
7986     \in@@#2#1\in@\in@@}
7987 \else
7988   \let\bbl@tempa\@empty
7989 \fi
7990 \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).

```
7991 \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.

```
7992 \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.

```
7993 \ifx\@tempcnta\@undefined
7994   \csname newcount\endcsname\@tempcnta\relax
7995 \fi
7996 \ifx\@tempcntb\@undefined
7997   \csname newcount\endcsname\@tempcntb\relax
7998 \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).

```
7999 \ifx\bye\@undefined
8000   \advance\count10 by -2\relax
8001 \fi
8002 \ifx\@ifnextchar\@undefined
8003   \def\@ifnextchar#1#2#3{%
8004     \let\reserved@d=#1%
8005     \def\reserved@a{#2}\def\reserved@b{#3}%
8006     \futurelet\@let@token\@ifnch}
8007   \def\@ifnch{%
8008     \ifx\@let@token\@sptoken
8009       \let\reserved@c\@xifnch
8010     \else
8011       \ifx\@let@token\reserved@d
8012         \let\reserved@c\reserved@a
8013       \else
8014         \let\reserved@c\reserved@b
8015       \fi
8016     \fi
8017     \reserved@c}
8018   \def\:{\let\@sptoken= } \:  % this makes \@sptoken a space token
8019   \def\:{\@xifnch} \expandafter\def\: {\futurelet\@let@token\@ifnch}
8020 \fi
8021 \def\@testopt#1#2{%
8022   \@ifnextchar[{#1}{#1[#2]}}
8023 \def\@protected@testopt#1{%
8024   \ifx\protect\@typeset@protect
8025     \expandafter\@testopt
8026   \else
8027     \@x@protect#1%
8028   \fi}
8029 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
8030       #2\relax}\fi}
8031 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
8032         \else\expandafter\@gobble\fi{#1}}
```

20.4   Encoding related macros

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

```
8033 \def\DeclareTextCommand{%
8034   \@dec@text@cmd\providecommand
8035 }
8036 \def\ProvideTextCommand{%
8037   \@dec@text@cmd\providecommand
8038 }
8039 \def\DeclareTextSymbol#1#2#3{%
8040   \@dec@text@cmd\chardef#1{#2}#3\relax
8041 }
8042 \def\@dec@text@cmd#1#2#3{%
8043   \expandafter\def\expandafter#2%
8044       \expandafter{%
```

```
8045              \csname#3-cmd\expandafter\endcsname
8046              \expandafter#2%
8047              \csname#3\string#2\endcsname
8048          }%
8049 %    \let\@ifdefinable\@rc@ifdefinable
8050      \expandafter#1\csname#3\string#2\endcsname
8051 }
8052 \def\@current@cmd#1{%
8053    \ifx\protect\@typeset@protect\else
8054        \noexpand#1\expandafter\@gobble
8055    \fi
8056 }
8057 \def\@changed@cmd#1#2{%
8058    \ifx\protect\@typeset@protect
8059        \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
8060            \expandafter\ifx\csname ?\string#1\endcsname\relax
8061              \expandafter\def\csname ?\string#1\endcsname{%
8062                  \@changed@x@err{#1}%
8063              }%
8064          \fi
8065          \global\expandafter\let
8066            \csname\cf@encoding \string#1\expandafter\endcsname
8067            \csname ?\string#1\endcsname
8068        \fi
8069        \csname\cf@encoding\string#1%
8070          \expandafter\endcsname
8071    \else
8072        \noexpand#1%
8073    \fi
8074 }
8075 \def\@changed@x@err#1{%
8076      \errhelp{Your command will be ignored, type <return> to proceed}%
8077      \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
8078 \def\DeclareTextCommandDefault#1{%
8079    \DeclareTextCommand#1?%
8080 }
8081 \def\ProvideTextCommandDefault#1{%
8082    \ProvideTextCommand#1?%
8083 }
8084 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
8085 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
8086 \def\DeclareTextAccent#1#2#3{%
8087  \DeclareTextCommand#1{#2}[1]{\accent#3 ##1}
8088 }
8089 \def\DeclareTextCompositeCommand#1#2#3#4{%
8090    \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
8091    \edef\reserved@b{\string##1}%
8092    \edef\reserved@c{%
8093      \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
8094    \ifx\reserved@b\reserved@c
8095      \expandafter\expandafter\expandafter\ifx
8096        \expandafter\@car\reserved@a\relax\relax\@nil
8097        \@text@composite
8098      \else
8099        \edef\reserved@b##1{%
8100            \def\expandafter\noexpand
8101              \csname#2\string#1\endcsname####1{%
8102              \noexpand\@text@composite
8103                \expandafter\noexpand\csname#2\string#1\endcsname
8104                ####1\noexpand\@empty\noexpand\@text@composite
8105                {##1}%
8106          }%
8107        }%
```

```
8108        \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
8109      \fi
8110      \expandafter\def\csname\expandafter\string\csname
8111         #2\endcsname\string#1-\string#3\endcsname{#4}
8112    \else
8113      \errhelp{Your command will be ignored, type <return> to proceed}%
8114      \errmessage{\string\DeclareTextCompositeCommand\space used on
8115         inappropriate command \protect#1}
8116    \fi
8117 }
8118 \def\@text@composite#1#2#3\@text@composite{%
8119    \expandafter\@text@composite@x
8120      \csname\string#1-\string#2\endcsname
8121 }
8122 \def\@text@composite@x#1#2{%
8123    \ifx#1\relax
8124        #2%
8125    \else
8126        #1%
8127    \fi
8128 }
8129 %
8130 \def\@strip@args#1:#2-#3\@strip@args{#2}
8131 \def\DeclareTextComposite#1#2#3#4{%
8132    \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
8133    \bgroup
8134        \lccode`\@=#4%
8135        \lowercase{%
8136    \egroup
8137        \reserved@a @%
8138    }%
8139 }
8140 %
8141 \def\UseTextSymbol#1#2{#2}
8142 \def\UseTextAccent#1#2#3{}
8143 \def\@use@text@encoding#1{}
8144 \def\DeclareTextSymbolDefault#1#2{%
8145    \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
8146 }
8147 \def\DeclareTextAccentDefault#1#2{%
8148    \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
8149 }
8150 \def\cf@encoding{OT1}
```

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

```
8151 \DeclareTextAccent{\"}{OT1}{127}
8152 \DeclareTextAccent{\'}{OT1}{19}
8153 \DeclareTextAccent{\^}{OT1}{94}
8154 \DeclareTextAccent{\`}{OT1}{18}
8155 \DeclareTextAccent{\~}{OT1}{126}
```

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

```
8156 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
8157 \DeclareTextSymbol{\textquotedblright}{OT1}{`\"}
8158 \DeclareTextSymbol{\textquoteleft}{OT1}{`\`}
8159 \DeclareTextSymbol{\textquoteright}{OT1}{`\'}
8160 \DeclareTextSymbol{\i}{OT1}{16}
8161 \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.

```
8162 \ifx\scriptsize\@undefined
8163  \let\scriptsize\sevenrm
```

219

```
8164 \fi
```

And a few more "dummy" definitions.

```
8165 \def\languagename{english}%
8166 \let\bbl@opt@shorthands\@nnil
8167 \def\bbl@ifshorthand#1#2#3{#2}%
8168 \let\bbl@language@opts\@empty
8169 \ifx\babeloptionstrings\@undefined
8170   \let\bbl@opt@strings\@nnil
8171 \else
8172   \let\bbl@opt@strings\babeloptionstrings
8173 \fi
8174 \def\BabelStringsDefault{generic}
8175 \def\bbl@tempa{normal}
8176 \ifx\babeloptionmath\bbl@tempa
8177   \def\bbl@mathnormal{\noexpand\textormath}
8178 \fi
8179 \def\AfterBabelLanguage#1#2{}
8180 \ifx\BabelModifiers\@undefined\let\BabelModifiers\relax\fi
8181 \let\bbl@afterlang\relax
8182 \def\bbl@opt@safe{BR}
8183 \ifx\@uclclist\@undefined\let\@uclclist\@empty\fi
8184 \ifx\bbl@trace\@undefined\def\bbl@trace#1{}\fi
8185 \expandafter\newif\csname ifbbl@single\endcsname
8186 \chardef\bbl@bidimode\z@
8187 ⟨⟨/Emulate LaTeX⟩⟩
```

A proxy file:

```
8188 ⟨∗plain⟩
8189 \input babel.def
8190 ⟨/plain⟩
```

21   Acknowledgements

I would like to thank all who volunteered as β-testers for their time. Michel Goossens supplied contributions for most of the other languages. Nico Poppelier helped polish the text of the documentation and supplied parts of the macros for the Dutch language. Paul Wackers and Werenfried Spit helped find and repair bugs.
During the further development of the babel system I received much help from Bernd Raichle, for which I am grateful.

References

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

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

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

[4] Donald E. Knuth, *The 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] Edward M. Reingold and Nachum Dershowitz, *Calendrical Calculations: The Ultimate Edition*, Cambridge University Press, 2018

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

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

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

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