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

Part I
User guide

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

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

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

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

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

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

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

1 The user interface

1.1 Monolingual documents

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

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

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

PDFTEX

```
\documentclass{article}

\usepackage[T1]{fontenc}
```

```
\usepackage[french]{babel}

\begin{document}

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

\end{document}
```

Now consider something like:

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

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

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

LUATEX/XETEX

```
\documentclass[russian]{article}

\usepackage{babel}

\babelfont{rm}{DejaVu Serif}

\begin{document}

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

\end{document}
```

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

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

Or the more explanatory:

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

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

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

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

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

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

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

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

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

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

NOTE Babel does not make any readjustments by default in font size, vertical positioning or line height by default. This is on purpose because the optimal solution depends on the document layout and the font, and very likely the most appropriate one is a combination of these settings.

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

7

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:

LUATEX/XETEX

```
\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.

New 3.84   With pdftex, when a language is loaded on the fly (actually, with \babelprovide) selectors now set the font encoding based on the list provided when loading fontenc. Not all scripts have an associated encoding, so this feature works only with Latin, Cyrillic, Greek, Arabic, Hebrew, Cherokee, Armenian, and Georgian, provided a suitable font is found.

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}.
```

---

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

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

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

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

1.6   Plain

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

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

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

1.7   Basic language selectors

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

\selectlanguage   {⟨*language*⟩}

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

```
\selectlanguage{german}
```

This command can be used as environment, too.

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

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.

---

[3]In old versions the error read "You haven't loaded the language LANG yet".

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}`).

`\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.

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

---

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

only work on 'known' shorthand characters, and an error will be raised otherwise. You can check if a character is a shorthand with \ifbabelshorthand (see below). New 3.9a However, \shorthandoff does not behave as you would expect with characters like ~ or ^, because they usually are not "other". For them \shorthandoff* is provided, so that with

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

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

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

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

\useshorthands ***** {⟨*char*⟩}

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

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

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

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

User shorthands override language ones, which in turn override system shorthands. Language-dependent user shorthands (new in 3.9) take precedence over "normal" user shorthands.

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

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

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

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

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

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

\languageshorthands {⟨*language*⟩}

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

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

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

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 " ^ =

---

[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.

[6]Thanks to Enrico Gregorio

14

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 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, `\aliasshorthands` 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.

---

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

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]

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;

---

[8]You can use alternatively the package silence.
[9]Turned off in plain.
[10]Duplicated options count as several ones.

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]

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.

---

[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.

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 უძ<ვ>ირესია 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]{1ດ 1ມ 1ອ 1ງ 1ກ 1ຈ} % 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]	ar-IQ	Arabic[u]
agq	Aghem	ar-JO	Arabic[u]
ak	Akan	ar-LB	Arabic[u]
am	Amharic[ul]	ar-MA	Arabic[u]
ar-DZ	Arabic[u]	ar-PS	Arabic[u]
ar-EG	Arabic[u]	ar-SA	Arabic[u]

ar-SY	Arabic[u]	en-NZ	English[ul]
ar-TN	Arabic[u]	en-US	American English[ul]
ar	Arabic[u]	en	English[ul]
as	Assamese[u]	eo	Esperanto[ul]
asa	Asu	es-MX	Mexican Spanish[ul]
ast	Asturian[ul]	es	Spanish[ul]
az-Cyrl	Azerbaijani	et	Estonian[ul]
az-Latn	Azerbaijani	eu	Basque[ull]
az	Azerbaijani[ul]	ewo	Ewondo
bas	Basaa	fa	Persian[u]
be	Belarusian[ul]	ff	Fulah
bem	Bemba	fi	Finnish[ul]
bez	Bena	fil	Filipino
bg	Bulgarian[ul]	fo	Faroese
bm	Bambara	fr-BE	French[ul]
bn	Bangla[u]	fr-CA	Canadian French[ul]
bo	Tibetan[u]	fr-CH	Swiss French[ul]
br	Breton[ul]	fr-LU	French[ul]
brx	Bodo	fr	French[ul]
bs-Cyrl	Bosnian	fur	Friulian[ul]
bs-Latn	Bosnian[ul]	fy	Western Frisian
bs	Bosnian[ul]	ga	Irish[ul]
ca	Catalan[ul]	gd	Scottish Gaelic[ul]
ce	Chechen	gl	Galician[ul]
cgg	Chiga	grc	Ancient Greek[ul]
chr	Cherokee	gsw	Swiss German
ckb-Arab	Central Kurdish[u]	gu	Gujarati
ckb-Latn	Central Kurdish[u]	guz	Gusii
ckb	Central Kurdish[u]	gv	Manx
cop	Coptic	ha-GH	Hausa
cs	Czech[ul]	ha-NE	Hausa
cu-Cyrs	Church Slavic[u]	ha	Hausa[ul]
cu-Glag	Church Slavic	haw	Hawaiian
cu	Church Slavic[u]	he	Hebrew[ul]
cy	Welsh[ul]	hi	Hindi[u]
da	Danish[ul]	hr	Croatian[ul]
dav	Taita	hsb	Upper Sorbian[ul]
de-1901	German[ul]	hu	Hungarian[ulll]
de-1996	German[ul]	hy	Armenian[ul]
de-AT-1901	Austrian German[ul]	ia	Interlingua[ul]
de-AT-1996	Austrian German[ul]	id	Indonesian[ul]
de-AT	Austrian German[ul]	ig	Igbo
de-CH-1901	Swiss High German[ul]	ii	Sichuan Yi
de-CH-1996	Swiss High 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[u]
dyo	Jola-Fonyi	kab	Kabyle
dz	Dzongkha	kam	Kamba
ebu	Embu	kde	Makonde
ee	Ewe	kea	Kabuverdianu
el-polyton	Polytonic Greek[ul]	kgp	Kaingang
el	Greek[ul]	khq	Koyra Chiini
en-AU	Australian English[ul]	ki	Kikuyu
en-CA	Canadian English[ul]	kk	Kazakh
en-GB	British English[ul]	kkj	Kako

Code	Language	Code	Language
kl	Kalaallisut	nus	Nuer
kln	Kalenjin	nyn	Nyankole
km	Khmer[u]	oc	Occitan[ul]
kmr-Arab	Northern Kurdish[u]	om	Oromo
kmr-Latn	Northern Kurdish[ul]	or	Odia
kmr	Northern Kurdish[ul]	os	Ossetic
kn	Kannada[u]	pa-Arab	Punjabi
ko-Hani	Korean[u]	pa-Guru	Punjabi[u]
ko	Korean[u]	pa	Punjabi[u]
kok	Konkani	pl	Polish[ul]
ks	Kashmiri	pms	Piedmontese[ul]
ksb	Shambala	ps	Pashto
ksf	Bafia	pt-BR	Brazilian Portuguese[ul]
ksh	Colognian	pt-PT	European Portuguese[ul]
kw	Cornish	pt	Portuguese[ul]
ky	Kyrgyz	qu	Quechua
la-x-classic	Classic Latin[ul]	rm	Romansh[ul]
la-x-ecclesia	Ecclesiastic Latin[ul]	rn	Rundi
la-x-medieval	Medieval Latin[ul]	ro-MD	Moldavian[ul]
la	Latin[ul]	ro	Romanian[ul]
lag	Langi	rof	Rombo
lb	Luxembourgish[ul]	ru	Russian[ul]
lg	Ganda	rw	Kinyarwanda
lkt	Lakota	rwk	Rwa
ln	Lingala	sa-Beng	Sanskrit
lo	Lao[u]	sa-Deva	Sanskrit
lrc	Northern Luri	sa-Gujr	Sanskrit
lt	Lithuanian[ulll]	sa-Knda	Sanskrit
lu	Luba-Katanga	sa-Mlym	Sanskrit
luo	Luo	sa-Telu	Sanskrit
luy	Luyia	sa	Sanskrit
lv	Latvian[ul]	sah	Sakha
mas	Masai	saq	Samburu
mer	Meru	sbp	Sangu
mfe	Morisyen	sc	Sardinian
mg	Malagasy	se	Northern Sami[ul]
mgh	Makhuwa-Meetto	seh	Sena
mgo	Meta'	ses	Koyraboro Senni
mk	Macedonian[ul]	sg	Sango
ml	Malayalam[u]	shi-Latn	Tachelhit
mn	Mongolian	shi-Tfng	Tachelhit
mr	Marathi[u]	shi	Tachelhit
ms-BN	Malay	si	Sinhala[u]
ms-SG	Malay	sk	Slovak[ul]
ms	Malay[ul]	sl	Slovenian[ul]
mt	Maltese	smn	Inari Sami
mua	Mundang	sn	Shona
my	Burmese	so	Somali
mzn	Mazanderani	sq	Albanian[ul]
naq	Nama	sr-Cyrl-BA	Serbian[ul]
nb	Norwegian Bokmål[ul]	sr-Cyrl-ME	Serbian[ul]
nd	North Ndebele	sr-Cyrl-XK	Serbian[ul]
ne	Nepali	sr-Cyrl	Serbian[ul]
nl	Dutch[ul]	sr-Latn-BA	Serbian[ul]
nmg	Kwasio	sr-Latn-ME	Serbian[ul]
nn	Norwegian Nynorsk[ul]	sr-Latn-XK	Serbian[ul]
nnh	Ngiemboon	sr-Latn	Serbian[ul]
no	Norwegian[ul]	sr	Serbian[ul]

sv	Swedish[ul]		vai	Vai
sw	Swahili		vi	Vietnamese[ul]
syr	Syriac		vun	Vunjo
ta	Tamil[u]		wae	Walser
te	Telugu[u]		xog	Soga
teo	Teso		yav	Yangben
th	Thai[ul]		yi	Yiddish
ti	Tigrinya		yo	Yoruba
tk	Turkmen[ul]		yrl	Nheengatu
to	Tongan		yue	Cantonese
tr	Turkish[ul]		zgh	Standard Moroccan Tamazight
twq	Tasawaq			
tzm	Central Atlas Tamazight		zh-Hans-HK	Chinese
ug	Uyghur[u]		zh-Hans-MO	Chinese
uk	Ukrainian[ul]		zh-Hans-SG	Chinese
ur	Urdu[u]		zh-Hans	Chinese[u]
uz-Arab	Uzbek		zh-Hant-HK	Chinese
uz-Cyrl	Uzbek		zh-Hant-MO	Chinese
uz-Latn	Uzbek		zh-Hant	Chinese[u]
uz	Uzbek		zh	Chinese[u]
vai-Latn	Vai		zu	Zulu
vai-Vaii	Vai			

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.

afrikaans	basaa
aghem	basque
akan	belarusian
albanian	bemba
american	bena
amharic	bangla
ancientgreek	bodo
arabic	bosnian-cyrillic
arabic-algeria	bosnian-cyrl
arabic-DZ	bosnian-latin
arabic-morocco	bosnian-latn
arabic-MA	bosnian
arabic-syria	brazilian
arabic-SY	breton
armenian	british
assamese	bulgarian
asturian	burmese
asu	canadian
australian	cantonese
austrian	catalan
azerbaijani-cyrillic	centralatlastamazight
azerbaijani-cyrl	centralkurdish
azerbaijani-latin	chechen
azerbaijani-latn	cherokee
azerbaijani	chiga
bafia	chinese-hans-hk
bambara	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

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

---

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

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	uyghur
swedish	uzbek-arab
swissgerman	uzbek-arabic
tachelhit-latin	uzbek-cyrillic
tachelhit-latn	uzbek-cyrl
tachelhit-tfng	uzbek-latin
tachelhit-tifinagh	uzbek-latn
tachelhit	uzbek
taita	vai-latin
tamil	vai-latn
tasawaq	vai-vai
telugu	vai-vaii
teso	vai
thai	vietnam
tibetan	vietnamese
tigrinya	vunjo
tongan	walser
turkish	welsh
turkmen	westernfrisian
ukenglish	yangben
ukrainian	yiddish
uppersorbian	yoruba
urdu	zarma
usenglish	zulu
usorbian	

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

---

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

many fonts as you want 'just in case', because if the language is never selected, the corresponding \babelfont declaration is just ignored.

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

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

LUATEX/XETEX
```
\documentclass{article}

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

\babelprovide[import]{hebrew}

\babelfont{rm}{FreeSerif}

\begin{document}

Svenska \foreignlanguage{hebrew}{עִבְרִית} svenska.

\end{document}
```

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

LUATEX/XETEX
```
\babelfont{rm}{Iwona}
\babelfont[hebrew]{rm}{FreeSerif}
```

\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:

LUATEX/XETEX
```
\babelfont{kai}{FandolKai}
```

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

NOTE You may load fontspec explicitly. For example:

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

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

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

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

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

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

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

> **This is *not* an error.** babel assumes that if you are using `\babelfont` for a family, very likely you want to define the rest of them. If you don't, you can find some inconsistencies between families. This checking is done at the beginning of the document, at a point where we cannot know which families will be used.
>
> Actually, there is no real need to use `\babelfont` in a monolingual document, if you set the language system in `\setmainfont` (or not, depending on what you want).
>
> As the message explains, *there is nothing intrinsically wrong* with not defining all the families. In fact, there is nothing intrinsically wrong with not using `\babelfont` at all. But you must be aware that this may lead to some problems.

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

1.15   Modifying a language

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

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

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

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

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

NOTE There are a few alternative methods:

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

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

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

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

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

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

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

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

This redefinition is immediate.

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

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

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

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

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

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

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

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

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

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

1.16 Creating a language

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

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

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

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

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

EXAMPLE If you need a language named `arhinish`:

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

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

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

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

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

import= ⟨*language-tag*⟩

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

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

Unicode engines load the UTF-8 variants, while 8-bit engines load the LICR (ie, with macros like `\'` or `\ss`) ones.
New 3.23 It may be used without a value, and that is often the recommended option. 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 is best written as:

```
\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 is an alternative to `justification=unhyphenated`.

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.polutonico]{babel}
```

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

```
\usepackage[italian, polytonicgreek, provide=*]{babel}
```

Remerber there is an alternative syntax for the latter:

```
\usepackage[italian]{babel}
\babelprovide[import, main]{polytonicgreek}
```

Finally, also remember you might not need to load `italian` at all if there are only a few word in this language (see 1.3).

script= ⟨*script-name*⟩

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

language= ⟨*language-name*⟩

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

alph= ⟨*counter-name*⟩

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

Alph= ⟨*counter-name*⟩

Same for `\Alph`.

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

onchar= ids | fonts | letters

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

> **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.

> **NOTE** There is no general rule to set the font for a punctuation mark, because it is a semantic decision and not a typographical one. Consider the following sentence: "یک, دو, and سه are Persian numbers". In this case the punctuation font must be the English one, even if the commas are surrounded by non-Latin letters. Quotation marks, parenthesis, etc., are even more complex. Several criteria are possible, like the main language (the default in babel), the first letter in the paragraph, or the surrounding letters, among others, but even so manual switching can be still necessary.

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

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

intrapenalty= ⟨*penalty*⟩

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

transforms= ⟨*transform-list*⟩

See section 1.21.

justification= unhyphenated | kashida | elongated | padding

New 3.59 There are currently 4 options. Note they are language dependent, so that they will not be applied to other languages.
The first one (unhyphenated) activates a line breaking mode that allows spaces to be stretched to arbitrary amounts. Although for European standards the result may look odd, in some writing systems, like Malayalam and other Indic scripts, this has been the customary (although not always the desired) practice. Because of that, no locale sets currently this mode by default (Amharic is an exception). Unlike `\sloppy`, the `\hfuzz` and the `\vfuzz` are not changed, because this line breaking mode is not really 'sloppy' (in other words, overfull boxes are reported as usual).

The second and the third are for the Arabic script. It sets the linebreaking and justification method, which can be based on the the ARABIC TATWEEL character or in the 'justification alternatives' OpenType table (jalt). For an explanation see the [babel site](#).

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

linebreaking= New 3.59 Just a synonymous for justification.

> **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*⟩}
\localecounter {⟨*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, syllable, 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 | ethiopic | islamic-civil | persian |
| coptic | hebrew | islamic-umalqura | |

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 \getlocaleproperty*, described below, is the preferred command, so that the existence of a field can be checked before. This also means building a string with the language and the region with \localeinfo*{language.tab.bcp47}-\localeinfo*{region.tab.bcp47} is not usually a good idea (because of the hyphen).

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

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

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

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

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

\LocaleForEach {⟨*code*⟩}

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

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

1.20 Hyphenation and line breaking

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

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

New 3.9a   It is customary to classify hyphens in two types: (1) *explicit* or *hard hyphens*, which in TeX are entered as -, and (2) *optional* or *soft hyphens*, which are entered as \-. Strictly, a *soft hyphen* is not a hyphen, but just a breaking opportunity or, in TeX terms, a "discretionary"; a *hard hyphen* is a hyphen with a breaking opportunity after it. A further type is a *non-breaking hyphen*, a hyphen without a breaking opportunity.

In TeX, - and \- forbid further breaking opportunities in the word. This is the desired behavior very often, but not always, and therefore many languages provide shorthands for these cases. Unfortunately, this has not been done consistently: for example, "- in Dutch, Portuguese, Catalan or Danish is a hard hyphen, while in German, Spanish, Norwegian, Slovak or Russian is a soft hyphen. Furthermore, some of them even redefine \-, so that you cannot insert a soft hyphen without breaking opportunities in the rest of the word. Therefore, some macros are provided with a set of basic "hyphens" which can be used by themselves, to define a user shorthand, or even in language files.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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:

---

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

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

```
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.
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, Odia, 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 ([ǐǔ]), the replacement could be {1|ǐǔ|íú}, which maps *ǐ* to *í*, and *ǔ* to *ú*, so that the diaeresis is removed.

This feature is activated with the first `\babelposthyphenation` or `\babelprehyphenation`. New 3.85 Another option is `label`, which takes a value similar to those in `\babelprovide` key `transforms` (in fact, the latter just applies this option). This label can be used to turn on and off transforms with a higher level interface, by means of `\enablelocaletransform` and `\disablelocaletransform` (see below).

New 3.85 When used in conjunction with `label`, this key makes a transform font dependent. As an example, the rules for Arabic kashida can differ depending on the font design. The value consists in a list of space-separated font tags:

```
\babelprehyphenation[label=transform.name, fonts=rm sf]{..}{..}
```

Tags can adopt two forms: a family, such as `rm` or `tt`, or the set family/series/shape. If a font matches one of these conditions, the transform is enabled. The second tag in `rm rm/n/it` is redundant. There are no wildcards; so, for italics you may want to write something like `sf/m/it sf/b/it`.

Transforms set for specific fonts (at least once in any language) are always reset with a font selector.

In `\babelprovide`, transform labels can be tagged before its name, with a list separated with colons, like:

```
transforms = rm:sf:transform.name
```

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.

`\enablelocaletransform` {⟨*label*⟩}
`\disablelocaletransform` {⟨*label*⟩}

New 3.85 Enables and disables the transform with the given label in the current language.

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 `\getlocaleproperty`. It must be turned on explicitly for similar reasons to those explained above.

1.23 Selecting scripts

Currently babel provides no standard interface to select scripts, because they are best selected with either `\fontencoding` (low-level) or a language name (high-level). Even the Latin script may require different encodings (ie, sets of glyphs) depending on the language, and therefore such a switch would be in a sense incomplete.[17]
Some languages sharing the same script define macros to switch it (eg, `\textcyrillic`), but be aware they may also set the language to a certain default. Even the babel core defined `\textlatin`, but is was somewhat buggy because in some cases it messed up encodings and fonts (for example, if the main Latin encoding was LY1), and therefore it has been deprecated.[18]

`\ensureascii` `{⟨text⟩}`

New 3.9i   This macro makes sure ⟨text⟩ is typeset with a LICR-savvy encoding in the ASCII range. It is used to redefine `\TeX` and `\LaTeX` so that they are correctly typeset even with LGR or X2 (the complete list is stored in `\BabelNonASCII`, which by default is LGR, X2, OT2, OT3, OT6, LHE, LWN, LMA, LMC, LMS, LMU, but you can modify it). So, in some sense it fixes the bug described in the previous paragraph.
If non-ASCII encodings are not loaded (or no encoding at all), it is no-op (also `\TeX` and `\LaTeX` are not redefined); otherwise, `\ensureascii` switches to the encoding at the beginning of the document if ASCII-savvy, or else the last ASCII-savvy encoding loaded. For example, if you load LY1,LGR, then it is set to LY1, but if you load LY1,T2A it is set to T2A. The symbol encodings TS1, T3, and TS3 are not taken into account, since they are not used for "ordinary" text (they are stored in `\BabelNonText`, used in some special cases when no Latin encoding is explicitly set).
The foregoing rules (which are applied "at begin document") cover most of the cases. No assumption is made on characters above 127, which may not follow the LICR conventions – the goal is just to ensure most of the ASCII letters and symbols are the right ones.

1.24 Selecting directions

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

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

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

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

---

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

[18]But still defined for backwards compatibility.

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

There are some package options controlling bidi writing.

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

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

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

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

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

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

```
\documentclass{article}

\usepackage[bidi=basic]{babel}

\babelprovide[import, main]{arabic}

\babelfont{rm}{FreeSerif}

\begin{document}

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

\end{document}
```

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

```
\documentclass{book}

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

\babelprovide[onchar=ids fonts]{arabic}

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

\begin{document}

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

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

    \end{document}
```

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

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

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

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

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

New 3.16 *To be expanded.* Selects which layout elements are adapted in bidi documents, including some text elements (except with options loading the `bidi` package, which provides its own mechanism to control these elements). You may use several options with a space-separated list, like `layout=counters contents sectioning` (in New 3.85 spaces are to be preferred over dots, which was the former syntax). 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.

New 3.84 Since `\thepage` is (indirectly) redefined, `makeindex` will reject many entries as invalid. With `counters*` babel attempts to remove the conflicting macros.

`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` .

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
```

\localerestoredirs

`New 3.86` *LuaTeX*. This command resets the internal text, paragraph and body directions to those of the current locale (if different). Sometimes changing directly these values can be useful for some hacks, and this command helps in restoring the directions to the correct ones. It can be used in > arguments of array, too.

\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}
```

47

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)[19]
Romanian romanian
Russian russian
Scottish Gaelic scottish
Spanish spanish

---

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

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

There are more languages not listed above, including hindi, thai, thaicjk, latvian, turkmen, magyar, mongolian, romansh, lithuanian, spanglish, vietnamese, japanese, pinyin, arabic, farsi, ibygreek, bgreek, serbianc, frenchle, ethiop and friulan.

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

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

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

1.28   Unicode character properties in luatex

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

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

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

For example:

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

Please, refer to the Unicode standard (Annex #9 and Annex #14) for the meaning of the available codes. For example, en is 'European number' and id is 'ideographic'.

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

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

1.29   Tweaking some features

\babeladjust {⟨*key-value-list*⟩}

New 3.36   Sometimes you might need to disable some babel features. Currently this macro understands the following keys [to be documented], with values on or off:

```
bidi.mirroring          linebreak.cjk          autoload.bcp47
bidi.text               justify.arabic         bcp47.toname
bidi.math               layout.tabular
linebreak.sea           layout.lists
```

Other keys [to be documented] are:

```
autoload.options        autoload.bcp47.options select.write
autoload.bcp47.prefix   prehyphenation.disable select.encoding
```

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.[20] 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.)

---

[20]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.

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.

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.[21]. 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,

---

[21]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.

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).[22] 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).[23]

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[24]. 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.[25] 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.

---

[22]This feature was added to 3.9o, but it was buggy. Both 3.9o and 3.9p are deprecated.
[23]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`.
[24]This is because different operating systems sometimes use *very* different file-naming conventions.
[25]This is not a new feature, but in former versions it didn't work correctly.

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

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

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

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

- The language definition files must define five macros, used to activate and deactivate the language-specific definitions. These macros are `\⟨lang⟩hyphenmins`, `\captions⟨lang⟩`, `\date⟨lang⟩`, `\extras⟨lang⟩` and `\noextras⟨lang⟩`(the last two may be left empty); where ⟨lang⟩ is either the name of the language definition file or the name of the LaTeX option that is to be used. These macros and their functions are 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, `\babel@save` and `\babel@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.[26]

---

[26]But not removed, for backward compatibility.

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

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

3.1 Guidelines for contributed languages

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

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

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

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

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

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

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

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

3.2 Basic macros

In the core of the babel system, several macros are defined for use in language definition files. Their purpose is 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]

\bbl@add@special
\bbl@remove@special
The TEXbook states: "Plain TEX includes a macro called \dospecials that is essentially a set macro, representing the set of all characters that have a special category code." [4, p. 380] It is used to set text 'verbatim'. To make this work if more characters get a special category code, you have to add this character to the macro \dospecial. 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.

\@safe@activestrue
\@safe@activesfalse
Enables and disables the "safe" mode. It is a tool for package and class authors. See the description below.

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[27].

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

\babel@savevariable
A second macro is provided to save the current value of a variable. In this context, anything that is allowed after the \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
\bbl@nonfrenchspacing
The commands \bbl@frenchspacing and \bbl@nonfrenchspacing can be used to properly switch French spacing on and off.

---

[27]This mechanism was introduced by Bernd Raichle.

3.8 Encoding-dependent strings

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

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

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

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

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

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

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

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

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

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

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

The ⟨*category*⟩ is either `captions`, `date` or `extras`. You must stick to these three categories, even if no error is raised when using other name.[28] 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}
```

---

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

```
\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}
  \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.[29]

\EndBabelCommands Marks the end of the series of blocks.

\AfterBabelCommands {⟨*code*⟩}

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

---

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

59

\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*⟩}

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=3.87.09667⟩⟩
 2 ⟨⟨date=2023/04/06⟩⟩
```

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

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

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

```
23    \fi}
24 \def\bbl@for#1#2#3{\bbl@loopx#1{#2}{\ifx#1\@empty\else#3\fi}}
```

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

```
25 \def\bbl@add@list#1#2{%
26   \edef#1{%
27     \bbl@ifunset{\bbl@stripslash#1}%
28       {}%
29       {\ifx#1\@empty\else#1,\fi}%
30     #2}}
```

\bbl@afterelse Because the code that is used in the handling of active characters may need to look ahead, we take
\bbl@afterfi extra care to 'throw' it over the \else and \fi parts of an \if-statement[30]. These macros will break if another \if...\fi statement appears in one of the arguments and it is not enclosed in braces.

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

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

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

\bbl@trim The following piece of code is stolen (with some changes) from keyval, by David Carlisle. It defines two macros: \bbl@trim and \bbl@trim@def. The first one strips the leading and trailing spaces from the second argument and then applies the first argument (a macro, \toks@ and the like). The second one, as its name suggests, defines the first argument as the stripped second argument.

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

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

```
56 \begingroup
57   \gdef\bbl@ifunset#1{%
58     \expandafter\ifx\csname#1\endcsname\relax
59       \expandafter\@firstoftwo
```

---

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

```
60      \else
61        \expandafter\@secondoftwo
62      \fi}
63    \bbl@ifunset{ifcsname}%
64      {}%
65      {\gdef\bbl@ifunset#1{%
66        \ifcsname#1\endcsname
67          \expandafter\ifx\csname#1\endcsname\relax
68            \bbl@afterelse\expandafter\@firstoftwo
69          \else
70            \bbl@afterfi\expandafter\@secondoftwo
71          \fi
72        \else
73          \expandafter\@firstoftwo
74        \fi}}
75  \endgroup
```

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

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

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

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

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

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

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

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

```
111    \expandafter\bbl@replace@aux#1#2\bbl@nil#2%
112    \edef#1{\the\toks@}}
```

An 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).

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

Two further tools. \bbl@ifsamestring first expand its arguments and then compare their expansion (sanitized, so that the catcodes do not matter). \bbl@engine takes the following values: 0 is pdfTeX, 1 is luatex, and 2 is xetex. You may use the latter it in your language style if you want.

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

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

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

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

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

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).

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

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

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

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

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

Start with some "private" debugging tool, and then define macros for errors.
```
211 \@ifpackagewith{babel}{debug}
212   {\providecommand\bbl@trace[1]{\message{^^J[ #1 ]}}%
213    \let\bbl@debug\@firstofone
214    \ifx\directlua\@undefined\else
215      \directlua{ Babel = Babel or {}
216        Babel.debug = true }%
217      \input{babel-debug.tex}%
218    \fi}
219   {\providecommand\bbl@trace[1]{}%
220    \let\bbl@debug\@gobble
221    \ifx\directlua\@undefined\else
222      \directlua{ Babel = Babel or {}
223        Babel.debug = false }%
224    \fi}
225 \def\bbl@error#1#2{%
226   \begingroup
227     \def\\{\MessageBreak}%
228     \PackageError{babel}{#1}{#2}%
229   \endgroup}
230 \def\bbl@warning#1{%
231   \begingroup
232     \def\\{\MessageBreak}%
233     \PackageWarning{babel}{#1}%
234   \endgroup}
235 \def\bbl@infowarn#1{%
236   \begingroup
237     \def\\{\MessageBreak}%
238     \PackageNote{babel}{#1}%
239   \endgroup}
240 \def\bbl@info#1{%
241   \begingroup
242     \def\\{\MessageBreak}%
243     \PackageInfo{babel}{#1}%
244   \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.
```
245 ⟨⟨Basic macros⟩⟩
```

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

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

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

```
292 \bbl@trace{key=value and another general options}
293 \bbl@csarg\let{tempa\expandafter}\csname opt@babel.sty\endcsname
294 \def\bbl@tempb#1.#2{%  Remove trailing dot
295    #1\ifx\@empty#2\else,\bbl@afterfi\bbl@tempb#2\fi}%
296 \def\bbl@tempd#1.#2\@nnil{%  TODO. Refactor lists?
297   \ifx\@empty#2%
298     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
299   \else
300     \in@{,provide=}{,#1}%
301     \ifin@
302       \edef\bbl@tempc{%
303         \ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.\bbl@tempb#2}%
304     \else
305       \in@{=}{#1}%
306       \ifin@
307         \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.#2}%
308       \else
309         \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
310         \bbl@csarg\edef{mod@#1}{\bbl@tempb#2}%
311       \fi
312     \fi
313   \fi}
314 \let\bbl@tempc\@empty
315 \bbl@foreach\bbl@tempa{\bbl@tempd#1.\@empty\@nnil}
316 \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.

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

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

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

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

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

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

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

```
362 \ProcessOptions*

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

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=....

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

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

```
389    \def\bbl@ifshorthand#1{%
390      \bbl@xin@{\string#1}{\bbl@opt@shorthands}%
391      \ifin@
392        \expandafter\@firstoftwo
```

```
393    \else
394      \expandafter\@secondoftwo
395    \fi}
```

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

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

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

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

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

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

```
413 \bbl@trace{Defining IfBabelLayout}
414 \ifx\bbl@opt@layout\@nnil
415   \newcommand\IfBabelLayout[3]{#3}%
416 \else
417   \bbl@exp{\\\bbl@forkv{\@nameuse{@raw@opt@babel.sty}}}{%
418      \in@{,layout,}{,#1,}%
419      \ifin@
420        \def\bbl@opt@layout{#2}%
421        \bbl@replace\bbl@opt@layout{ }{.}%
422      \fi}
423   \newcommand\IfBabelLayout[1]{%
424      \@expandtwoargs\in@{.#1.}{.\bbl@opt@layout.}%
425      \ifin@
426        \expandafter\@firstoftwo
427      \else
428        \expandafter\@secondoftwo
429      \fi}
430 \fi
431 ⟨/package⟩
432 ⟨∗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*.

```
433 \ifx\ldf@quit\@undefined\else
434 \endinput\fi % Same line!
```

435 ⟨⟨*Make sure ProvidesFile is defined*⟩⟩
436 \ProvidesFile{babel.def}[⟨⟨*date*⟩⟩ ⟨⟨*version*⟩⟩ Babel common definitions]
437 \ifx\AtBeginDocument\@undefined % TODO. change test.
438   ⟨⟨*Emulate LaTeX*⟩⟩
439 \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.

440 ⟨/core⟩
441 ⟨*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.

442 \def\bbl@version{⟨⟨*version*⟩⟩}
443 \def\bbl@date{⟨⟨*date*⟩⟩}
444 ⟨⟨*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.

445 \def\adddialect#1#2{%
446   \global\chardef#1#2\relax
447   \bbl@usehooks{adddialect}{{#1}{#2}}%
448   \begingroup
449   \count@#1\relax
450   \def\bbl@elt##1##2##3##4{%
451   \ifnum\count@=##2\relax
452   \edef\bbl@tempa{\expandafter\@gobbletwo\string#1}%
453   \bbl@info{Hyphen rules for '\expandafter\@gobble\bbl@tempa'
454   set to \expandafter\string\csname l@##1\endcsname\\%
455   (\string\language\the\count@). Reported}%
456   \def\bbl@elt####1####2####3####4{}%
457   \fi}%
458   \bbl@cs{languages}%
459   \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.

460 \def\bbl@fixname#1{%
461   \begingroup
462   \def\bbl@tempe{l@}%
463   \edef\bbl@tempd{\noexpand\@ifundefined{\noexpand\bbl@tempe#1}}%
464   \bbl@tempd
465   {\lowercase\expandafter{\bbl@tempd}%
466   {\uppercase\expandafter{\bbl@tempd}%
467   \@empty
468   {\edef\bbl@tempd{\def\noexpand#1{#1}}%
469   \uppercase\expandafter{\bbl@tempd}}}%
470   {\edef\bbl@tempd{\def\noexpand#1{#1}}%
471   \lowercase\expandafter{\bbl@tempd}}}%
472   \@empty
473   \edef\bbl@tempd{\endgroup\def\noexpand#1{#1}}%
474   \bbl@tempd
475   \bbl@exp{\\\bbl@usehooks{languagename}{{\languagename}{#1}}}}
476 \def\bbl@iflanguage#1{%
477   \@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.

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

```
536        \edef\localename{\bbl@bcp@prefix\bbl@bcp}%
537        \expandafter\ifx\csname date\languagename\endcsname\relax
538          \let\bbl@initoload\bbl@bcp
539          \bbl@exp{\\\babelprovide[\bbl@autoload@bcpoptions]{\languagename}}%
540          \let\bbl@initoload\relax
541        \fi
542        \bbl@csarg\xdef{bcp@map@\bbl@bcp}{\localename}%
543      \fi
544    \fi
545  \fi
546  \expandafter\ifx\csname date\languagename\endcsname\relax
547    \IfFileExists{babel-\languagename.tex}%
548      {\bbl@exp{\\\babelprovide[\bbl@autoload@options]{\languagename}}}%
549      {}%
550  \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.

```
551 \def\iflanguage#1{%
552   \bbl@iflanguage{#1}{%
553     \ifnum\csname l@#1\endcsname=\language
554       \expandafter\@firstoftwo
555     \else
556       \expandafter\@secondoftwo
557     \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.

```
558 \let\bbl@select@type\z@
559 \edef\selectlanguage{%
560   \noexpand\protect
561   \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.

```
562 \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.

```
563 \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.

```
564 \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
\bbl@pop@language

The stack is simply a list of languagenames, separated with a '+' sign; the push function can be simple:

```
565 \def\bbl@push@language{%
566   \ifx\languagename\@undefined\else
567     \ifx\currentgrouplevel\@undefined
568       \xdef\bbl@language@stack{\languagename+\bbl@language@stack}%
569     \else
570       \ifnum\currentgrouplevel=\z@
571         \xdef\bbl@language@stack{\languagename+}%
572       \else
573         \xdef\bbl@language@stack{\languagename+\bbl@language@stack}%
574       \fi
575     \fi
576   \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.

```
577 \def\bbl@pop@lang#1+#2\@@{%
578   \edef\languagename{#1}%
579   \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).

```
580 \let\bbl@ifrestoring\@secondoftwo
581 \def\bbl@pop@language{%
582   \expandafter\bbl@pop@lang\bbl@language@stack\@@
583   \let\bbl@ifrestoring\@firstoftwo
584   \expandafter\bbl@set@language\expandafter{\languagename}%
585   \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).

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

The unprotected part of \selectlanguage.

```
604 \expandafter\def\csname selectlanguage \endcsname#1{%
```

```
605    \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\tw@\fi
606    \bbl@push@language
607    \aftergroup\bbl@pop@language
608    \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).

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

```
655     \def\bbl@selectorname{select}%
656   % set hymap
657   \fi
658   \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
659   % set name
660   \edef\languagename{#1}%
661   \bbl@fixname\languagename
662   % TODO. name@map must be here?
663   \bbl@provide@locale
664   \bbl@iflanguage\languagename{%
665     \let\bbl@select@type\z@
666     \expandafter\bbl@switch\expandafter{\languagename}}}
667 \def\babel@aux#1#2{%
668   \select@language{#1}%
669   \bbl@foreach\BabelContentsFiles{% \relax -> don't assume vertical mode
670     \@writefile{##1}{\babel@toc{#1}{#2}\relax}}}% TODO - plain?
671 \def\babel@toc#1#2{%
672   \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.

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

```
705    \bbl@esphack
706    % switch extras
707    \csname bbl@preextras@#1\endcsname
708    \bbl@usehooks{beforeextras}{}%
709    \csname extras#1\endcsname\relax
710    \bbl@usehooks{afterextras}{}%
711    %  > babel-ensure
712    %  > babel-sh-<short>
713    %  > babel-bidi
714    %  > babel-fontspec
715    \let\bbl@savedextras\@empty
716    % hyphenation - case mapping
717    \ifcase\bbl@opt@hyphenmap\or
718      \def\BabelLower##1##2{\lccode##1=##2\relax}%
719      \ifnum\bbl@hymapsel>4\else
720        \csname\languagename @bbl@hyphenmap\endcsname
721      \fi
722      \chardef\bbl@opt@hyphenmap\z@
723    \else
724      \ifnum\bbl@hymapsel>\bbl@opt@hyphenmap\else
725        \csname\languagename @bbl@hyphenmap\endcsname
726      \fi
727    \fi
728    \let\bbl@hymapsel\@cclv
729    % hyphenation - select rules
730    \ifnum\csname l@\languagename\endcsname=\l@unhyphenated
731      \edef\bbl@tempa{u}%
732    \else
733      \edef\bbl@tempa{\bbl@cl{lnbrk}}%
734    \fi
735    % linebreaking - handle u, e, k (v in the future)
736    \bbl@xin@{/u}{/\bbl@tempa}%
737    \ifin@\else\bbl@xin@{/e}{/\bbl@tempa}\fi % elongated forms
738    \ifin@\else\bbl@xin@{/k}{/\bbl@tempa}\fi % only kashida
739    \ifin@\else\bbl@xin@{/p}{/\bbl@tempa}\fi % padding (eg, Tibetan)
740    \ifin@\else\bbl@xin@{/v}{/\bbl@tempa}\fi % variable font
741    \ifin@
742      % unhyphenated/kashida/elongated/padding = allow stretching
743      \language\l@unhyphenated
744      \babel@savevariable\emergencystretch
745      \emergencystretch\maxdimen
746      \babel@savevariable\hbadness
747      \hbadness\@M
748    \else
749      % other = select patterns
750      \bbl@patterns{#1}%
751    \fi
752    % hyphenation - mins
753    \babel@savevariable\lefthyphenmin
754    \babel@savevariable\righthyphenmin
755    \expandafter\ifx\csname #1hyphenmins\endcsname\relax
756      \set@hyphenmins\tw@\thr@@\relax
757    \else
758      \expandafter\expandafter\expandafter\set@hyphenmins
759        \csname #1hyphenmins\endcsname\relax
760    \fi
761    \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.

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

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

```
767 \long\def\endotherlanguage{%
768   \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.

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

```
776 \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.

```
777 \providecommand\bbl@beforeforeign{}
778 \edef\foreignlanguage{%
779   \noexpand\protect
780   \expandafter\noexpand\csname foreignlanguage \endcsname}
781 \expandafter\def\csname foreignlanguage \endcsname{%
782   \@ifstar\bbl@foreign@s\bbl@foreign@x}
783 \providecommand\bbl@foreign@x[3][]{%
784   \begingroup
785     \def\bbl@selectorname{foreign}%
786     \def\bbl@select@opts{#1}%
787     \let\BabelText\@firstofone
788     \bbl@beforeforeign
789     \foreign@language{#2}%
790     \bbl@usehooks{foreign}{}%
791     \BabelText{#3}% Now in horizontal mode!
```

```
792    \endgroup}
793 \def\bbl@foreign@s#1#2{% TODO - \shapemode, \@setpar, ?\@@par
794    \begingroup
795      {\par}%
796      \def\bbl@selectorname{foreign*}%
797      \let\bbl@select@opts\@empty
798      \let\BabelText\@firstofone
799      \foreign@language{#1}%
800      \bbl@usehooks{foreign*}{}%
801      \bbl@dirparastext
802      \BabelText{#2}% Still in vertical mode!
803      {\par}%
804    \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.

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

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

```
818 \def\IfBabelSelectorTF#1{%
819    \bbl@xin@{,\bbl@selectorname,}{,\zap@space#1 \@empty,}%
820    \ifin@
821      \expandafter\@firstoftwo
822    \else
823      \expandafter\@secondoftwo
824    \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.

```
825 \let\bbl@hyphlist\@empty
826 \let\bbl@hyphenation@\relax
827 \let\bbl@pttnlist\@empty
828 \let\bbl@patterns@\relax
829 \let\bbl@hymapsel=\@cclv
830 \def\bbl@patterns#1{%
831    \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
832      \csname l@#1\endcsname
833      \edef\bbl@tempa{#1}%
834    \else
835      \csname l@#1:\f@encoding\endcsname
836      \edef\bbl@tempa{#1:\f@encoding}%
837    \fi
838    \@expandtwoargs\bbl@usehooks{patterns}{{#1}{\bbl@tempa}}%
839    % > luatex
```

```
840    \@ifundefined{bbl@hyphenation@}{}{% Can be \relax!
841      \begingroup
842        \bbl@xin@{,\number\language,}{,\bbl@hyphlist}%
843        \ifin@\else
844          \@expandtwoargs\bbl@usehooks{hyphenation}{{#1}{\bbl@tempa}}%
845          \hyphenation{%
846            \bbl@hyphenation@
847            \@ifundefined{bbl@hyphenation@#1}%
848              \@empty
849              {\space\csname bbl@hyphenation@#1\endcsname}}%
850          \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
851        \fi
852      \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*.

```
853 \def\hyphenrules#1{%
854   \edef\bbl@tempf{#1}%
855   \bbl@fixname\bbl@tempf
856   \bbl@iflanguage\bbl@tempf{%
857     \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
858     \ifx\languageshorthands\@undefined\else
859       \languageshorthands{none}%
860     \fi
861     \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax
862       \set@hyphenmins\tw@\thr@@\relax
863     \else
864       \expandafter\expandafter\expandafter\set@hyphenmins
865       \csname\bbl@tempf hyphenmins\endcsname\relax
866     \fi}}
867 \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.

```
868 \def\providehyphenmins#1#2{%
869   \expandafter\ifx\csname #1hyphenmins\endcsname\relax
870     \@namedef{#1hyphenmins}{#2}%
871   \fi}
```

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

```
872 \def\set@hyphenmins#1#2{%
873   \lefthyphenmin#1\relax
874   \righthyphenmin#2\relax}
```

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

```
875 \ifx\ProvidesFile\@undefined
876   \def\ProvidesLanguage#1[#2 #3 #4]{%
877     \wlog{Language: #1 #4 #3 <#2>}%
878     }
879 \else
880   \def\ProvidesLanguage#1{%
881     \begingroup
882       \catcode`\ 10 %
883       \@makeother\/%
884       \@ifnextchar[%
```

```
885        {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}}
886   \def\@provideslanguage#1[#2]{%
887     \wlog{Language: #1 #2}%
888     \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
889     \endgroup}
890 \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.

```
891 \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.

```
892 \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':

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

7.2 Errors

\@nolanerr   The babel package will signal an error when a documents tries to select a language that hasn't been
\@nopatterns defined earlier. When a user selects a language for which no hyphenation patterns were loaded into
the format he will be given a warning about that fact. We revert to the patterns for \language=0 in
that case. In most formats that will be (US)english, but it might also be empty.

\@noopterr When the package was loaded without options not everything will work as expected. An error
message is issued in that case.
When the format knows about \PackageError it must be LATEX 2ε, 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.

```
903 \edef\bbl@nulllanguage{\string\language=0}
904 \def\bbl@nocaption{\protect\bbl@nocaption@i}
905 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
906   \global\@namedef{#2}{\textbf{?#1?}}%
907   \@nameuse{#2}%
908   \edef\bbl@tempa{#1}%
909   \bbl@sreplace\bbl@tempa{name}{}%
910   \bbl@warning{%
911     \@backslashchar#1 not set for '\languagename'. Please,\\%
912     define it after the language has been loaded\\%
913     (typically in the preamble) with:\\%
914     \string\setlocalecaption{\languagename}{\bbl@tempa}{..}\\%
915     Feel free to contribute on github.com/latex3/babel.\\%
916     Reported}}
917 \def\bbl@tentative{\protect\bbl@tentative@i}
918 \def\bbl@tentative@i#1{%
919   \bbl@warning{%
920     Some functions for '#1' are tentative.\\%
921     They might not work as expected and their behavior\\%
922     could change in the future.\\%
923     Reported}}
924 \def\@nolanerr#1{%
925   \bbl@error
```

```
926      {You haven't defined the language '#1' yet.\\%
927       Perhaps you misspelled it or your installation\\%
928       is not complete}%
929      {Your command will be ignored, type <return> to proceed}}
930 \def\@nopatterns#1{%
931   \bbl@warning
932     {No hyphenation patterns were preloaded for\\%
933      the language '#1' into the format.\\%
934      Please, configure your TeX system to add them and\\%
935      rebuild the format. Now I will use the patterns\\%
936      preloaded for \bbl@nulllanguage\space instead}}
937 \let\bbl@usehooks\@gobbletwo
938 \ifx\bbl@onlyswitch\@empty\endinput\fi
939   % Here ended switch.def
```

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

```
940 \ifx\directlua\@undefined\else
941   \ifx\bbl@luapatterns\@undefined
942     \input luababel.def
943   \fi
944 \fi
945 ⟨⟨Basic macros⟩⟩
946 \bbl@trace{Compatibility with language.def}
947 \ifx\bbl@languages\@undefined
948   \ifx\directlua\@undefined
949     \openin1 = language.def % TODO. Remove hardcoded number
950     \ifeof1
951       \closein1
952       \message{I couldn't find the file language.def}
953     \else
954       \closein1
955       \begingroup
956         \def\addlanguage#1#2#3#4#5{%
957           \expandafter\ifx\csname lang@#1\endcsname\relax\else
958             \global\expandafter\let\csname l@#1\expandafter\endcsname
959               \csname lang@#1\endcsname
960           \fi}%
961         \def\uselanguage#1{}%
962         \input language.def
963       \endgroup
964     \fi
965   \fi
966   \chardef\l@english\z@
967 \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.

```
968 \def\addto#1#2{%
969   \ifx#1\@undefined
970     \def#1{#2}%
971   \else
972     \ifx#1\relax
973       \def#1{#2}%
974     \else
975       {\toks@\expandafter{#1#2}%
976        \xdef#1{\the\toks@}}%
977     \fi
978   \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.

```
979 \def\bbl@withactive#1#2{%
980   \begingroup
981     \lccode`\~=`#2\relax
982     \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.

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

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

```
988 \def\bbl@redefine@long#1{%
989   \edef\bbl@tempa{\bbl@stripslash#1}%
990   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
991   \long\expandafter\def\csname\bbl@tempa\endcsname}
992 \@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␣.

```
993 \def\bbl@redefinerobust#1{%
994   \edef\bbl@tempa{\bbl@stripslash#1}%
995   \bbl@ifunset{\bbl@tempa\space}%
996     {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
997      \bbl@exp{\def\\#1{\\\protect\<\bbl@tempa\space>}}}%
998     {\bbl@exp{\let\<org@\bbl@tempa>\<\bbl@tempa\space>}}%
999   \@namedef{\bbl@tempa\space}}
1000 \@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.

```
1001 \bbl@trace{Hooks}
1002 \newcommand\AddBabelHook[3][]{%
1003   \bbl@ifunset{bbl@hk@#2}{\EnableBabelHook{#2}}{}%
1004   \def\bbl@tempa##1,#3=##2,##3\@empty{\def\bbl@tempb{##2}}%
1005   \expandafter\bbl@tempa\bbl@evargs,#3=,\@empty
1006   \bbl@ifunset{bbl@ev@#2@#3@#1}%
1007     {\bbl@csarg\bbl@add{ev@#3@#1}{\bbl@elth{#2}}}%
1008     {\bbl@csarg\let{ev@#2@#3@#1}\relax}%
1009   \bbl@csarg\newcommand{ev@#2@#3@#1}[\bbl@tempb]}
1010 \newcommand\EnableBabelHook[1]{\bbl@csarg\let{hk@#1}\@firstofone}
1011 \newcommand\DisableBabelHook[1]{\bbl@csarg\let{hk@#1}\@gobble}
1012 \def\bbl@usehooks{\bbl@usehooks@lang\languagename}
1013 \def\bbl@usehooks@lang#1#2#3{%
1014   \ifx\UseHook\@undefined\else\UseHook{babel/*/#2}\fi
1015   \def\bbl@elth##1{%
1016     \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#2@}#3}}%
1017   \bbl@cs{ev@#2@}%
1018   \ifx\languagename\@undefined\else % Test required for Plain (?)
1019     \ifx\UseHook\@undefined\else\UseHook{babel/#1/#2}\fi
1020     \def\bbl@elth##1{%
1021       \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#2@#1}#3}}%
1022     \bbl@cs{ev@#2@#1}%
1023   \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).

```
1024 \def\bbl@evargs{,% <- don't delete this comma
1025   everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
1026   adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
1027   beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
1028   hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%
1029   beforestart=0,languagename=2,begindocument=1}
1030 \ifx\NewHook\@undefined\else
1031   \def\bbl@tempa#1=#2\@@{\NewHook{babel/#1}}
1032   \bbl@foreach\bbl@evargs{\bbl@tempa#1\@@}
1033 \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.

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

```
1074              ########1}}}}%
1075          {}%
1076        \toks@\expandafter{##1}%
1077        \edef##1{%
1078          \bbl@csarg\noexpand{ensure@\languagename}%
1079          {\the\toks@}}%
1080      \fi
1081      \expandafter\bbl@tempb
1082    \fi}%
1083  \expandafter\bbl@tempb\bbl@captionslist\today\@empty
1084  \def\bbl@tempa##1{% elt for include list
1085    \ifx##1\@empty\else
1086      \bbl@csarg\in@{ensure@\languagename\expandafter}\expandafter{##1}%
1087      \ifin@\else
1088        \bbl@tempb##1\@empty
1089      \fi
1090      \expandafter\bbl@tempa
1091    \fi}%
1092  \bbl@tempa#1\@empty}
1093 \def\bbl@captionslist{%
1094    \prefacename\refname\abstractname\bibname\chaptername\appendixname
1095    \contentsname\listfigurename\listtablename\indexname\figurename
1096    \tablename\partname\enclname\ccname\headtoname\pagename\seename
1097    \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.

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

```
1116    \ifx#2\@undefined\else
1117       \ldf@quit{#1}%
1118    \fi
1119  \else
1120    \expandafter\ifx\csname#2\endcsname\relax\else
1121       \ldf@quit{#1}%
1122    \fi
1123  \fi
1124  \bbl@ldfinit}
```

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

```
1125 \def\ldf@quit#1{%
1126   \expandafter\main@language\expandafter{#1}%
1127   \catcode`\@=\atcatcode \let\atcatcode\relax
1128   \catcode`\==\eqcatcode \let\eqcatcode\relax
1129   \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.

```
1130 \def\bbl@afterldf#1{% TODO. Merge into the next macro? Unused elsewhere
1131   \bbl@afterlang
1132   \let\bbl@afterlang\relax
1133   \let\BabelModifiers\relax
1134   \let\bbl@screset\relax}%
1135 \def\ldf@finish#1{%
1136   \loadlocalcfg{#1}%
1137   \bbl@afterldf{#1}%
1138   \expandafter\main@language\expandafter{#1}%
1139   \catcode`\@=\atcatcode \let\atcatcode\relax
1140   \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.

```
1141 \@onlypreamble\LdfInit
1142 \@onlypreamble\ldf@quit
1143 \@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.

```
1144 \def\main@language#1{%
1145   \def\bbl@main@language{#1}%
1146   \let\languagename\bbl@main@language % TODO. Set localename
1147   \bbl@id@assign
1148   \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.

```
1149 \def\bbl@beforestart{%
1150   \def\@nolanerr##1{%
1151     \bbl@warning{Undefined language '##1' in aux.\\Reported}}%
1152   \bbl@usehooks{beforestart}{}%
1153   \global\let\bbl@beforestart\relax}
1154 \AtBeginDocument{%
1155   {\@nameuse{bbl@beforestart}}%  Group!
1156   \if@filesw
1157     \providecommand\babel@aux[2]{}%
1158     \immediate\write\@mainaux{%
1159       \string\providecommand\string\babel@aux[2]{}}%
```

```
1160        \immediate\write\@mainaux{\string\@nameuse{bbl@beforestart}}%
1161    \fi
1162    \expandafter\selectlanguage\expandafter{\bbl@main@language}%
1163    \ifbbl@single  % must go after the line above.
1164        \renewcommand\selectlanguage[1]{}%
1165        \renewcommand\foreignlanguage[2]{#2}%
1166        \global\let\babel@aux\@gobbletwo  % Also as flag
1167    \fi}
1168 \ifcase\bbl@engine\or
1169    \AtBeginDocument{\pagedir\bodydir} % TODO - a better place
1170 \fi
```

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

```
1171 \def\select@language@x#1{%
1172    \ifcase\bbl@select@type
1173        \bbl@ifsamestring\languagename{#1}{}{\select@language{#1}}%
1174    \else
1175        \select@language{#1}%
1176    \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.

```
1177 \bbl@trace{Shorhands}
1178 \def\bbl@add@special#1{% 1:a macro like \", \?, etc.
1179    \bbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
1180    \bbl@ifunset{@sanitize}{}{\bbl@add\@sanitize{\@makeother#1}}%
1181    \ifx\nfss@catcodes\@undefined\else % TODO - same for above
1182        \begingroup
1183            \catcode`#1\active
1184            \nfss@catcodes
1185            \ifnum\catcode`#1=\active
1186                \endgroup
1187                \bbl@add\nfss@catcodes{\@makeother#1}%
1188            \else
1189                \endgroup
1190            \fi
1191    \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.

```
1192 \def\bbl@remove@special#1{%
1193    \begingroup
1194        \def\x##1##2{\ifnum`#1=`##2\noexpand\@empty
1195                    \else\noexpand##1\noexpand##2\fi}%
1196        \def\do{\x\do}%
1197        \def\@makeother{\x\@makeother}%
1198    \edef\x{\endgroup
1199        \def\noexpand\dospecials{\dospecials}%
1200        \expandafter\ifx\csname @sanitize\endcsname\relax\else
1201            \def\noexpand\@sanitize{\@sanitize}%
1202        \fi}%
1203    \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).

```
1204 \def\bbl@active@def#1#2#3#4{%
1205   \@namedef{#3#1}{%
1206     \expandafter\ifx\csname#2@sh@#1@\endcsname\relax
1207       \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
1208     \else
1209       \bbl@afterfi\csname#2@sh@#1@\endcsname
1210     \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.

```
1211   \long\@namedef{#3@arg#1}##1{%
1212     \expandafter\ifx\csname#2@sh@#1@\string##1@\endcsname\relax
1213       \bbl@afterelse\csname#4#1\endcsname##1%
1214     \else
1215       \bbl@afterfi\csname#2@sh@#1@\string##1@\endcsname
1216     \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.

```
1217 \def\initiate@active@char#1{%
1218   \bbl@ifunset{active@char\string#1}%
1219     {\bbl@withactive
1220       {\expandafter\@initiate@active@char\expandafter}#1\string#1#1}%
1221     {}}
```

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).

```
1222 \def\@initiate@active@char#1#2#3{%
1223   \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
1224   \ifx#1\@undefined
1225     \bbl@csarg\def{oridef@#2}{\def#1{\active@prefix#1\@undefined}}%
1226   \else
1227     \bbl@csarg\let{oridef@@#2}#1%
1228     \bbl@csarg\edef{oridef@#2}{%
1229       \let\noexpand#1%
1230       \expandafter\noexpand\csname bbl@oridef@@#2\endcsname}%
1231   \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*).

```
1232   \ifx#1#3\relax
1233     \expandafter\let\csname normal@char#2\endcsname#3%
1234   \else
```

```
1235     \bbl@info{Making #2 an active character}%
1236     \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
1237       \@namedef{normal@char#2}{%
1238         \textormath{#3}{\csname bbl@oridef@@#2\endcsname}}%
1239     \else
1240       \@namedef{normal@char#2}{#3}%
1241     \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).

```
1242     \bbl@restoreactive{#2}%
1243     \AtBeginDocument{%
1244       \catcode`#2\active
1245       \if@filesw
1246         \immediate\write\@mainaux{\catcode`\string#2\active}%
1247       \fi}%
1248     \expandafter\bbl@add@special\csname#2\endcsname
1249     \catcode`#2\active
1250   \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⟩).

```
1251   \let\bbl@tempa\@firstoftwo
1252   \if\string^#2%
1253     \def\bbl@tempa{\noexpand\textormath}%
1254   \else
1255     \ifx\bbl@mathnormal\@undefined\else
1256       \let\bbl@tempa\bbl@mathnormal
1257     \fi
1258   \fi
1259   \expandafter\edef\csname active@char#2\endcsname{%
1260     \bbl@tempa
1261       {\noexpand\if@safe@actives
1262          \noexpand\expandafter
1263          \expandafter\noexpand\csname normal@char#2\endcsname
1264        \noexpand\else
1265          \noexpand\expandafter
1266          \expandafter\noexpand\csname bbl@doactive#2\endcsname
1267        \noexpand\fi}%
1268     {\expandafter\noexpand\csname normal@char#2\endcsname}}%
1269   \bbl@csarg\edef{doactive#2}{%
1270     \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

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

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

```
1271   \bbl@csarg\edef{active@#2}{%
1272     \noexpand\active@prefix\noexpand#1%
1273     \expandafter\noexpand\csname active@char#2\endcsname}%
1274   \bbl@csarg\edef{normal@#2}{%
1275     \noexpand\active@prefix\noexpand#1%
1276     \expandafter\noexpand\csname normal@char#2\endcsname}%
1277   \bbl@ncarg\let#1{bbl@normal@#2}%
```

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.

```
1278    \bbl@active@def#2\user@group{user@active}{language@active}%
1279    \bbl@active@def#2\language@group{language@active}{system@active}%
1280    \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.

```
1281    \expandafter\edef\csname\user@group @sh@#2@@\endcsname
1282        {\expandafter\noexpand\csname normal@char#2\endcsname}%
1283    \expandafter\edef\csname\user@group @sh@#2@\string\protect@\endcsname
1284        {\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.

```
1285    \if\string'#2%
1286        \let\prim@s\bbl@prim@s
1287        \let\active@math@prime#1%
1288    \fi
1289    \bbl@usehooks{initiateactive}{{#1}{#2}{#3}}}
```

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

```
1290 ⟨⟨*More package options⟩⟩ ≡
1291 \DeclareOption{math=active}{}
1292 \DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
1293 ⟨⟨/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.

```
1294 \@ifpackagewith{babel}{KeepShorthandsActive}%
1295    {\let\bbl@restoreactive\@gobble}%
1296    {\def\bbl@restoreactive#1{%
1297        \bbl@exp{%
1298            \\\AfterBabelLanguage\\\CurrentOption
1299                {\catcode`#1=\the\catcode`#1\relax}%
1300            \\\AtEndOfPackage
1301                {\catcode`#1=\the\catcode`#1\relax}}}%
1302    \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.

```
1303 \def\bbl@sh@select#1#2{%
1304    \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
1305        \bbl@afterelse\bbl@scndcs
1306    \else
1307        \bbl@afterfi\csname#1@sh@#2@sel\endcsname
1308    \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.

```
1309 \begingroup
1310 \bbl@ifunset{ifincsname}% TODO. Ugly. Correct? Only Plain?
1311   {\gdef\active@prefix#1{%
1312     \ifx\protect\@typeset@protect
1313     \else
1314       \ifx\protect\@unexpandable@protect
1315         \noexpand#1%
1316       \else
1317         \protect#1%
1318       \fi
1319       \expandafter\@gobble
1320     \fi}}
1321   {\gdef\active@prefix#1{%
1322     \ifincsname
1323       \string#1%
1324       \expandafter\@gobble
1325     \else
1326       \ifx\protect\@typeset@protect
1327       \else
1328         \ifx\protect\@unexpandable@protect
1329           \noexpand#1%
1330         \else
1331           \protect#1%
1332         \fi
1333         \expandafter\expandafter\expandafter\@gobble
1334       \fi
1335     \fi}}
1336 \endgroup
```

\if@safe@actives In some circumstances it is necessary to be able to reset the shorthand to its 'normal' value (usually
the character with catcode 'other') on the fly. For this purpose the switch @safe@actives is available.
The setting of this switch should be checked in the first level expansion of \active@char$\langle char\rangle$.
When this expansion mode is active (with \@safe@activestrue), something like $"_{13}"_{13}$ becomes
$"_{12}"_{12}$ in an \edef (in other words, shorthands are \string'ed). This contrasts with
\protected@edef, where catcodes are always left unchanged. Once converted, they can be used
safely even after this expansion mode is deactivated (with \@safe@activefalse).

```
1337 \newif\if@safe@actives
1338 \@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.

```
1339 \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$\langle char\rangle$ in the case of \bbl@activate, or
\normal@char$\langle char\rangle$ in the case of \bbl@deactivate.

```
1340 \chardef\bbl@activated\z@
1341 \def\bbl@activate#1{%
1342   \chardef\bbl@activated\@ne
1343   \bbl@withactive{\expandafter\let\expandafter}#1%
1344     \csname bbl@active@\string#1\endcsname}
1345 \def\bbl@deactivate#1{%
1346   \chardef\bbl@activated\tw@
1347   \bbl@withactive{\expandafter\let\expandafter}#1%
1348     \csname bbl@normal@\string#1\endcsname}
```

\bbl@firstcs These macros are used only as a trick when declaring shorthands.
\bbl@scndcs
```
1349 \def\bbl@firstcs#1#2{\csname#1\endcsname}
1350 \def\bbl@scndcs#1#2{\csname#2\endcsname}
```

The command \declare@shorthand is used to declare a shorthand on a certain level. It takes three arguments:

1. a name for the collection of shorthands, i.e. 'system', or 'dutch';

2. the character (sequence) that makes up the shorthand, i.e. ~ or "a;

3. the code to be executed when the shorthand is encountered.

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

```
1351 \def\babel@texpdf#1#2#3#4{%
1352   \ifx\texorpdfstring\@undefined
1353     \textormath{#1}{#3}%
1354   \else
1355     \texorpdfstring{\textormath{#1}{#3}}{#2}%
1356     % \texorpdfstring{\textormath{#1}{#3}}{\textormath{#2}{#4}}%
1357   \fi}
1358 %
1359 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
1360 \def\@decl@short#1#2#3\@nil#4{%
1361   \def\bbl@tempa{#3}%
1362   \ifx\bbl@tempa\@empty
1363     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
1364     \bbl@ifunset{#1@sh@\string#2@}{}%
1365       {\def\bbl@tempa{#4}%
1366        \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbl@tempa
1367        \else
1368          \bbl@info
1369            {Redefining #1 shorthand \string#2\\%
1370             in language \CurrentOption}%
1371        \fi}%
1372     \@namedef{#1@sh@\string#2@}{#4}%
1373   \else
1374     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs
1375     \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%
1376       {\def\bbl@tempa{#4}%
1377        \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
1378        \else
1379          \bbl@info
1380            {Redefining #1 shorthand \string#2\string#3\\%
1381             in language \CurrentOption}%
1382        \fi}%
1383     \@namedef{#1@sh@\string#2@\string#3@}{#4}%
1384   \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.

```
1385 \def\textormath{%
1386   \ifmmode
1387     \expandafter\@secondoftwo
1388   \else
1389     \expandafter\@firstoftwo
1390   \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'.

```
1391 \def\user@group{user}
1392 \def\language@group{english} % TODO. I don't like defaults
1393 \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. |

```
1394 \def\useshorthands{%
1395   \@ifstar\bbl@usesh@s{\bbl@usesh@x{}}}
1396 \def\bbl@usesh@s#1{%
1397   \bbl@usesh@x
1398     {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bbl@activate{#1}}}%
1399     {#1}}
1400 \def\bbl@usesh@x#1#2{%
1401   \bbl@ifshorthand{#2}%
1402     {\def\user@group{user}%
1403      \initiate@active@char{#2}%
1404      #1%
1405      \bbl@activate{#2}}%
1406     {\bbl@error
1407       {I can't declare a shorthand turned off (\string#2)}
1408       {Sorry, but you can't use shorthands which have been\\%
1409        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. |

```
1410 \def\user@language@group{user@\language@group}
1411 \def\bbl@set@user@generic#1#2{%
1412   \bbl@ifunset{user@generic@active#1}%
1413     {\bbl@active@def#1\user@language@group{user@active}{user@generic@active}%
1414      \bbl@active@def#1\user@group{user@generic@active}{language@active}%
1415      \expandafter\edef\csname#2@sh@#1@@\endcsname{%
1416        \expandafter\noexpand\csname normal@char#1\endcsname}%
1417      \expandafter\edef\csname#2@sh@#1@\string\protect@\endcsname{%
1418        \expandafter\noexpand\csname user@active#1\endcsname}}%
1419   \@empty}
1420 \newcommand\defineshorthand[3][user]{%
1421   \edef\bbl@tempa{\zap@space#1 \@empty}%
1422   \bbl@for\bbl@tempb\bbl@tempa{%
1423     \if*\expandafter\@car\bbl@tempb\@nil
1424       \edef\bbl@tempb{user@\expandafter\@gobble\bbl@tempb}%
1425       \@expandtwoargs
1426         \bbl@set@user@generic{\expandafter\string\@car#2\@nil}\bbl@tempb
1427     \fi
1428     \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]. |

```
1429 \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". |

```
1430 \def\aliasshorthand#1#2{%
1431   \bbl@ifshorthand{#2}%
1432     {\expandafter\ifx\csname active@char\string#2\endcsname\relax
1433        \ifx\document\@notprerr
1434          \@notshorthand{#2}%
1435        \else
1436          \initiate@active@char{#2}%
1437          \bbl@ccarg\let{active@char\string#2}{active@char\string#1}%
1438          \bbl@ccarg\let{normal@char\string#2}{normal@char\string#1}%
```

94

```
1439          \bbl@activate{#2}%
1440        \fi
1441      \fi}%
1442    {\bbl@error
1443        {Cannot declare a shorthand turned off (\string#2)}
1444        {Sorry, but you cannot use shorthands which have been\\%
1445         turned off in the package options}}}
```

\@notshorthand

```
1446 \def\@notshorthand#1{%
1447   \bbl@error{%
1448     The character '\string #1' should be made a shorthand character;\\%
1449     add the command \string\useshorthands\string{#1\string} to
1450     the preamble.\\%
1451     I will ignore your instruction}%
1452    {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.

```
1453 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
1454 \DeclareRobustCommand*\shorthandoff{%
1455   \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
1456 \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.

```
1457 \def\bbl@switch@sh#1#2{%
1458   \ifx#2\@nnil\else
1459     \bbl@ifunset{bbl@active@\string#2}%
1460       {\bbl@error
1461         {I can't switch '\string#2' on or off--not a shorthand}%
1462         {This character is not a shorthand. Maybe you made\\%
1463          a typing mistake? I will ignore your instruction.}}%
1464     {\ifcase#1%   off, on, off*
1465         \catcode`#212\relax
1466       \or
1467         \catcode`#2\active
1468         \bbl@ifunset{bbl@shdef@\string#2}%
1469           {}%
1470           {\bbl@withactive{\expandafter\let\expandafter}#2%
1471              \csname bbl@shdef@\string#2\endcsname
1472            \bbl@csarg\let{shdef@\string#2}\relax}%
1473         \ifcase\bbl@activated\or
1474           \bbl@activate{#2}%
1475         \else
1476           \bbl@deactivate{#2}%
1477         \fi
1478       \or
1479         \bbl@ifunset{bbl@shdef@\string#2}%
1480           {\bbl@withactive{\bbl@csarg\let{shdef@\string#2}}#2}%
1481           {}%
1482         \csname bbl@oricat@\string#2\endcsname
1483         \csname bbl@oridef@\string#2\endcsname
1484       \fi}%
1485     \bbl@afterfi\bbl@switch@sh#1%
1486   \fi}
```

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

```
1487 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
1488 \def\bbl@putsh#1{%
1489   \bbl@ifunset{bbl@active@\string#1}%
1490     {\bbl@putsh@i#1\@empty\@nnil}%
1491     {\csname bbl@active@\string#1\endcsname}}
1492 \def\bbl@putsh@i#1#2\@nnil{%
1493   \csname\language@group @sh@\string#1@%
1494     \ifx\@empty#2\else\string#2@\fi\endcsname}
1495 %
1496 \ifx\bbl@opt@shorthands\@nnil\else
1497   \let\bbl@s@initiate@active@char\initiate@active@char
1498   \def\initiate@active@char#1{%
1499     \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
1500   \let\bbl@s@switch@sh\bbl@switch@sh
1501   \def\bbl@switch@sh#1#2{%
1502     \ifx#2\@nnil\else
1503       \bbl@afterfi
1504       \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
1505     \fi}
1506   \let\bbl@s@activate\bbl@activate
1507   \def\bbl@activate#1{%
1508     \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
1509   \let\bbl@s@deactivate\bbl@deactivate
1510   \def\bbl@deactivate#1{%
1511     \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
1512 \fi
```

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

```
1513 \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.

```
1514 \def\bbl@prim@s{%
1515   \prime\futurelet\@let@token\bbl@pr@m@s}
1516 \def\bbl@if@primes#1#2{%
1517   \ifx#1\@let@token
1518     \expandafter\@firstoftwo
1519   \else\ifx#2\@let@token
1520     \bbl@afterelse\expandafter\@firstoftwo
1521   \else
1522     \bbl@afterfi\expandafter\@secondoftwo
1523   \fi\fi}
1524 \begingroup
1525   \catcode`\^=7  \catcode`\*=\active  \lccode`\*=`\^
1526   \catcode`\'=12 \catcode`\"=\active  \lccode`\"=`\'
1527   \lowercase{%
1528     \gdef\bbl@pr@m@s{%
1529       \bbl@if@primes"'%
1530         \pr@@@s
1531         {\bbl@if@primes*^\pr@@@t\egroup}}}
1532 \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).

```
1533 \initiate@active@char{~}
1534 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
1535 \bbl@activate{~}
```

96

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

```
1536 \expandafter\def\csname OT1dqpos\endcsname{127}
1537 \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

```
1538 \ifx\f@encoding\@undefined
1539   \def\f@encoding{OT1}
1540 \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.

```
1541 \bbl@trace{Language attributes}
1542 \newcommand\languageattribute[2]{%
1543   \def\bbl@tempc{#1}%
1544   \bbl@fixname\bbl@tempc
1545   \bbl@iflanguage\bbl@tempc{%
1546     \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.

```
1547       \ifx\bbl@known@attribs\@undefined
1548         \in@false
1549       \else
1550         \bbl@xin@{,\bbl@tempc-##1,}{,\bbl@known@attribs,}%
1551       \fi
1552       \ifin@
1553         \bbl@warning{%
1554           You have more than once selected the attribute '##1'\\%
1555           for language #1. Reported}%
1556       \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.

```
1557         \bbl@exp{%
1558           \\\bbl@add@list\\\bbl@known@attribs{\bbl@tempc-##1}}%
1559         \edef\bbl@tempa{\bbl@tempc-##1}%
1560         \expandafter\bbl@ifknown@ttrib\expandafter{\bbl@tempa}\bbl@attributes%
1561         {\csname\bbl@tempc @attr@##1\endcsname}%
1562         {\@attrerr{\bbl@tempc}{##1}}%
1563     \fi}}}
1564 \@onlypreamble\languageattribute
```

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

```
1565 \newcommand*{\@attrerr}[2]{%
1566   \bbl@error
1567     {The attribute #2 is unknown for language #1.}%
1568     {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}.

```
1569 \def\bbl@declare@ttribute#1#2#3{%
1570   \bbl@xin@{,#2,}{,\BabelModifiers,}%
```

```
1571        \ifin@
1572          \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
1573        \fi
1574      \bbl@add@list\bbl@attributes{#1-#2}%
1575      \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.

```
1576 \def\bbl@ifattributeset#1#2#3#4{%
1577    \ifx\bbl@known@attribs\@undefined
1578      \in@false
1579    \else
1580      \bbl@xin@{,#1-#2,}{,\bbl@known@attribs,}%
1581    \fi
1582    \ifin@
1583      \bbl@afterelse#3%
1584    \else
1585      \bbl@afterfi#4%
1586    \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.

```
1587 \def\bbl@ifknown@ttrib#1#2{%
1588    \let\bbl@tempa\@secondoftwo
1589    \bbl@loopx\bbl@tempb{#2}{%
1590      \expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{,#1,}%
1591      \ifin@
1592        \let\bbl@tempa\@firstoftwo
1593      \else
1594      \fi}%
1595    \bbl@tempa}
```

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

```
1596 \def\bbl@clear@ttribs{%
1597    \ifx\bbl@attributes\@undefined\else
1598      \bbl@loopx\bbl@tempa{\bbl@attributes}{%
1599        \expandafter\bbl@clear@ttrib\bbl@tempa.}%
1600      \let\bbl@attributes\@undefined
1601    \fi}
1602 \def\bbl@clear@ttrib#1-#2.{%
1603    \expandafter\let\csname#1@attr@#2\endcsname\@undefined}
1604 \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
```
1605 \bbl@trace{Macros for saving definitions}
1606 \def\babel@beginsave{\babel@savecnt\z@}
```

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

```
1607 \newcount\babel@savecnt
1608 \babel@beginsave
```

\babel@save The macro \babel@save⟨csname⟩ saves the current meaning of the control sequence ⟨csname⟩ to
\babel@savevariable \originalTeX[31]. 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. To avoid messing saved definitions up, they are saved only the very first
time.

```
1609 \def\babel@save#1{%
1610   \def\bbl@tempa{{,#1,}}% Clumsy, for Plain
1611   \expandafter\bbl@add\expandafter\bbl@tempa\expandafter{%
1612     \expandafter{\expandafter,\bbl@savedextras,}}%
1613   \expandafter\in@\bbl@tempa
1614   \ifin@\else
1615     \bbl@add\bbl@savedextras{,#1,}%
1616     \bbl@carg\let{babel@\number\babel@savecnt}#1\relax
1617     \toks@\expandafter{\originalTeX\let#1=}%
1618     \bbl@exp{%
1619       \def\\\originalTeX{\the\toks@\<babel@\number\babel@savecnt>\relax}}%
1620     \advance\babel@savecnt\@ne
1621   \fi}
1622 \def\babel@savevariable#1{%
1623   \toks@\expandafter{\originalTeX #1=}%
1624   \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.

```
1625 \def\bbl@frenchspacing{%
1626   \ifnum\the\sfcode`\.=\@m
1627     \let\bbl@nonfrenchspacing\relax
1628   \else
1629     \frenchspacing
1630     \let\bbl@nonfrenchspacing\nonfrenchspacing
1631   \fi}
1632 \let\bbl@nonfrenchspacing\nonfrenchspacing
1633 \let\bbl@elt\relax
1634 \edef\bbl@fs@chars{%
1635   \bbl@elt{\string.}\@m{3000}\bbl@elt{\string?}\@m{3000}%
1636   \bbl@elt{\string!}\@m{3000}\bbl@elt{\string:}\@m{2000}%
1637   \bbl@elt{\string;}\@m{1500}\bbl@elt{\string,}\@m{1250}}
1638 \def\bbl@pre@fs{%
1639   \def\bbl@elt##1##2##3{\sfcode`##1=\the\sfcode`##1\relax}%
1640   \edef\bbl@save@sfcodes{\bbl@fs@chars}}
1641 \def\bbl@post@fs{%
1642   \bbl@save@sfcodes
1643   \edef\bbl@tempa{\bbl@cl{frspc}}%
1644   \edef\bbl@tempa{\expandafter\@car\bbl@tempa\@nil}%
1645   \if u\bbl@tempa          % do nothing
1646   \else\if n\bbl@tempa     % non french
1647     \def\bbl@elt##1##2##3{%
1648       \ifnum\sfcode`##1=##2\relax
1649         \babel@savevariable{\sfcode`##1}%
1650         \sfcode`##1=##3\relax
1651       \fi}%
1652     \bbl@fs@chars
1653   \else\if y\bbl@tempa      % french
```

---

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

99

```
1654      \def\bbl@elt##1##2##3{%
1655        \ifnum\sfcode`##1=##3\relax
1656          \babel@savevariable{\sfcode`##1}%
1657          \sfcode`##1=##2\relax
1658        \fi}%
1659      \bbl@fs@chars
1660    \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.

```
1661 \bbl@trace{Short tags}
1662 \def\babeltags#1{%
1663    \edef\bbl@tempa{\zap@space#1 \@empty}%
1664    \def\bbl@tempb##1=##2\@@{%
1665       \edef\bbl@tempc{%
1666          \noexpand\newcommand
1667          \expandafter\noexpand\csname ##1\endcsname{%
1668             \noexpand\protect
1669             \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}%
1670          \noexpand\newcommand
1671          \expandafter\noexpand\csname text##1\endcsname{%
1672             \noexpand\foreignlanguage{##2}}}%
1673       \bbl@tempc}%
1674    \bbl@for\bbl@tempa\bbl@tempa{%
1675       \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.

```
1676 \bbl@trace{Hyphens}
1677 \@onlypreamble\babelhyphenation
1678 \AtEndOfPackage{%
1679    \newcommand\babelhyphenation[2][\@empty]{%
1680       \ifx\bbl@hyphenation@\relax
1681          \let\bbl@hyphenation@\@empty
1682       \fi
1683       \ifx\bbl@hyphlist\@empty\else
1684          \bbl@warning{%
1685             You must not intermingle \string\selectlanguage\space and\\%
1686             \string\babelhyphenation\space or some exceptions will not\\%
1687             be taken into account. Reported}%
1688       \fi
1689       \ifx\@empty#1%
1690          \protected@edef\bbl@hyphenation@{\bbl@hyphenation@\space#2}%
1691       \else
1692          \bbl@vforeach{#1}{%
1693             \def\bbl@tempa{##1}%
1694             \bbl@fixname\bbl@tempa
1695             \bbl@iflanguage\bbl@tempa{%
1696                \bbl@csarg\protected@edef{hyphenation@\bbl@tempa}{%
1697                   \bbl@ifunset{bbl@hyphenation@\bbl@tempa}%
1698                      {}%
1699                      {\csname bbl@hyphenation@\bbl@tempa\endcsname\space}%
1700                   #2}}}%
1701       \fi}}
```

\bbl@allowhyphens   This macro makes hyphenation possible. Basically its definition is nothing more than `\nobreak` `\hskip 0pt plus 0pt`[32].

```
1702 \def\bbl@allowhyphens{\ifvmode\else\nobreak\hskip\z@skip\fi}
1703 \def\bbl@t@one{T1}
1704 \def\allowhyphens{\ifx\cf@encoding\bbl@t@one\else\bbl@allowhyphens\fi}
```

\babelhyphen   Macros to insert common hyphens. Note the space before @ in `\babelhyphen`. Instead of protecting it with `\DeclareRobustCommand`, which could insert a `\relax`, we use the same procedure as shorthands, with `\active@prefix`.

```
1705 \newcommand\babelnullhyphen{\char\hyphenchar\font}
1706 \def\babelhyphen{\active@prefix\babelhyphen\bbl@hyphen}
1707 \def\bbl@hyphen{%
1708   \@ifstar{\bbl@hyphen@i @}{\bbl@hyphen@i\@empty}}
1709 \def\bbl@hyphen@i#1#2{%
1710   \bbl@ifunset{bbl@hy@#1#2\@empty}%
1711     {\csname bbl@#1usehyphen\endcsname{\discretionary{#2}{}{#2}}}%
1712     {\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.

```
1713 \def\bbl@usehyphen#1{%
1714   \leavevmode
1715   \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
1716   \nobreak\hskip\z@skip}
1717 \def\bbl@@usehyphen#1{%
1718   \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}
```

The following macro inserts the hyphen char.

```
1719 \def\bbl@hyphenchar{%
1720   \ifnum\hyphenchar\font=\m@ne
1721     \babelnullhyphen
1722   \else
1723     \char\hyphenchar\font
1724   \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.

```
1725 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}{}}}
1726 \def\bbl@hy@@soft{\bbl@@usehyphen{\discretionary{\bbl@hyphenchar}{}{}}}
1727 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
1728 \def\bbl@hy@@hard{\bbl@@usehyphen\bbl@hyphenchar}
1729 \def\bbl@hy@nobreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}}
1730 \def\bbl@hy@@nobreak{\mbox{\bbl@hyphenchar}}
1731 \def\bbl@hy@repeat{%
1732   \bbl@usehyphen{%
1733     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
1734 \def\bbl@hy@@repeat{%
1735   \bbl@@usehyphen{%
1736     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
1737 \def\bbl@hy@empty{\hskip\z@skip}
1738 \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.

```
1739 \def\bbl@disc#1#2{\nobreak\discretionary{#2-}{}{#1}\bbl@allowhyphens}
```

---

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

7.10 Multiencoding strings

The aim following commands is to provide a commom interface for strings in several encodings. They also contains several hooks which can be used by luatex and xetex. The code is organized here with pseudo-guards, so we start with the basic commands.

Tools But first, a tool. It makes global a local variable. This is not the best solution, but it works.

```
1740 \bbl@trace{Multiencoding strings}
1741 \def\bbl@toglobal#1{\global\let#1#1}
```

The second one. We need to patch `\@uclclist`, but it is done once and only if `\SetCase` is used or if strings are encoded. The code is far from satisfactory for several reasons, including the fact `\@uclclist` is not a list any more. Therefore a package option is added to ignore it. Instead of gobbling the macro getting the next two elements (usually `\reserved@a`), we pass it as argument to `\bbl@uclc`. The parser is restarted inside `\⟨lang⟩@bbl@uclc` because we do not know how many expansions are necessary (depends on whether strings are encoded). The last part is tricky – when uppercasing, we have:

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

and starts over (and similarly when lowercasing).

```
1742 \@ifpackagewith{babel}{nocase}%
1743   {\let\bbl@patchuclc\relax}%
1744   {\def\bbl@patchuclc{%
1745     \global\let\bbl@patchuclc\relax
1746     \g@addto@macro\@uclclist{\reserved@b{\reserved@b\bbl@uclc}}%
1747     \gdef\bbl@uclc##1{%
1748       \let\bbl@encoded\bbl@encoded@uclc
1749       \bbl@ifunset{\languagename @bbl@uclc}% and resumes it
1750         {##1}%
1751         {\let\bbl@tempa##1\relax % Used by LANG@bbl@uclc
1752          \csname\languagename @bbl@uclc\endcsname}%
1753       {\bbl@tolower\@empty}{\bbl@toupper\@empty}}%
1754     \gdef\bbl@tolower{\csname\languagename @bbl@lc\endcsname}%
1755     \gdef\bbl@toupper{\csname\languagename @bbl@uc\endcsname}}}
1756 ⟨⟨*More package options⟩⟩ ≡
1757 \DeclareOption{nocase}{}
1758 ⟨⟨/More package options⟩⟩
```

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

```
1759 ⟨⟨*More package options⟩⟩ ≡
1760 \let\bbl@opt@strings\@nnil % accept strings=value
1761 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
1762 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
1763 \def\BabelStringsDefault{generic}
1764 ⟨⟨/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.

```
1765 \@onlypreamble\StartBabelCommands
1766 \def\StartBabelCommands{%
1767   \begingroup
1768   \@tempcnta="7F
1769   \def\bbl@tempa{%
1770     \ifnum\@tempcnta>"FF\else
1771       \catcode\@tempcnta=11
1772       \advance\@tempcnta\@ne
1773       \expandafter\bbl@tempa
1774     \fi}%
1775   \bbl@tempa
1776   ⟨⟨Macros local to BabelCommands⟩⟩
```

```
1777    \def\bbl@provstring##1##2{%
1778      \providecommand##1{##2}%
1779      \bbl@toglobal##1}%
1780    \global\let\bbl@scafter\@empty
1781    \let\StartBabelCommands\bbl@startcmds
1782    \ifx\BabelLanguages\relax
1783        \let\BabelLanguages\CurrentOption
1784    \fi
1785    \begingroup
1786    \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
1787    \StartBabelCommands}
1788 \def\bbl@startcmds{%
1789    \ifx\bbl@screset\@nnil\else
1790        \bbl@usehooks{stopcommands}{}%
1791    \fi
1792    \endgroup
1793    \begingroup
1794    \@ifstar
1795        {\ifx\bbl@opt@strings\@nnil
1796            \let\bbl@opt@strings\BabelStringsDefault
1797          \fi
1798        \bbl@startcmds@i}%
1799        \bbl@startcmds@i}
1800 \def\bbl@startcmds@i#1#2{%
1801    \edef\bbl@L{\zap@space#1 \@empty}%
1802    \edef\bbl@G{\zap@space#2 \@empty}%
1803    \bbl@startcmds@ii}
1804 \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.

```
1805 \newcommand\bbl@startcmds@ii[1][\@empty]{%
1806    \let\SetString\@gobbletwo
1807    \let\bbl@stringdef\@gobbletwo
1808    \let\AfterBabelCommands\@gobble
1809    \ifx\@empty#1%
1810        \def\bbl@sc@label{generic}%
1811        \def\bbl@encstring##1##2{%
1812            \ProvideTextCommandDefault##1{##2}%
1813            \bbl@toglobal##1%
1814            \expandafter\bbl@toglobal\csname\string?\string##1\endcsname}%
1815        \let\bbl@sctest\in@true
1816    \else
1817        \let\bbl@sc@charset\space % <- zapped below
1818        \let\bbl@sc@fontenc\space % <-   "       "
1819        \def\bbl@tempa##1=##2\@nil{%
1820            \bbl@csarg\edef{sc@\zap@space##1 \@empty}{##2 }}%
1821        \bbl@vforeach{label=#1}{\bbl@tempa##1\@nil}%
1822        \def\bbl@tempa##1 ##2{% space -> comma
1823            ##1%
1824            \ifx\@empty##2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa##2\fi}%
1825        \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%
1826        \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%
1827        \edef\bbl@sc@charset{\expandafter\zap@space\bbl@sc@charset\@empty}%
1828        \def\bbl@encstring##1##2{%
1829            \bbl@foreach\bbl@sc@fontenc{%
```

```
1830        \bbl@ifunset{T@####1}%
1831          {}%
1832          {\ProvideTextCommand##1{####1}{##2}%
1833           \bbl@toglobal##1%
1834           \expandafter
1835           \bbl@toglobal\csname####1\string##1\endcsname}}%
1836    \def\bbl@sctest{%
1837      \bbl@xin@{,\bbl@opt@strings,}{,\bbl@sc@label,\bbl@sc@fontenc,}}%
1838  \fi
1839  \ifx\bbl@opt@strings\@nnil        % ie, no strings key -> defaults
1840  \else\ifx\bbl@opt@strings\relax    % ie, strings=encoded
1841    \let\AfterBabelCommands\bbl@aftercmds
1842    \let\SetString\bbl@setstring
1843    \let\bbl@stringdef\bbl@encstring
1844  \else        % ie, strings=value
1845  \bbl@sctest
1846  \ifin@
1847    \let\AfterBabelCommands\bbl@aftercmds
1848    \let\SetString\bbl@setstring
1849    \let\bbl@stringdef\bbl@provstring
1850  \fi\fi\fi
1851  \bbl@scswitch
1852  \ifx\bbl@G\@empty
1853    \def\SetString##1##2{%
1854      \bbl@error{Missing group for string \string##1}%
1855        {You must assign strings to some category, typically\\%
1856         captions or extras, but you set none}}%
1857  \fi
1858  \ifx\@empty#1%
1859    \bbl@usehooks{defaultcommands}{}%
1860  \else
1861    \@expandtwoargs
1862    \bbl@usehooks{encodedcommands}{{\bbl@sc@charset}{\bbl@sc@fontenc}}%
1863  \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) .

```
1864  \def\bbl@forlang#1#2{%
1865    \bbl@for#1\bbl@L{%
1866      \bbl@xin@{,#1,}{,\BabelLanguages,}%
1867      \ifin@#2\relax\fi}}
1868  \def\bbl@scswitch{%
1869    \bbl@forlang\bbl@tempa{%
1870      \ifx\bbl@G\@empty\else
1871        \ifx\SetString\@gobbletwo\else
1872          \edef\bbl@GL{\bbl@G\bbl@tempa}%
1873          \bbl@xin@{,\bbl@GL,}{,\bbl@screset,}%
1874          \ifin@\else
1875            \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
1876            \xdef\bbl@screset{\bbl@screset,\bbl@GL}%
1877          \fi
1878        \fi
1879      \fi}}
1880  \AtEndOfPackage{%
1881    \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{}{#2}}}%
1882    \let\bbl@scswitch\relax}
1883  \@onlypreamble\EndBabelCommands
```

```
1884 \def\EndBabelCommands{%
1885   \bbl@usehooks{stopcommands}{}%
1886   \endgroup
1887   \endgroup
1888   \bbl@scafter}
1889 \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.

```
1890 \def\bbl@setstring#1#2{% eg, \prefacename{<string>}
1891   \bbl@forlang\bbl@tempa{%
1892     \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
1893     \bbl@ifunset{\bbl@LC}% eg, \germanchaptername
1894       {\bbl@exp{%
1895         \global\\\bbl@add\<\bbl@G\bbl@tempa>{\\\bbl@scset\\#1\<\bbl@LC>}}}%
1896       {}%
1897     \def\BabelString{#2}%
1898     \bbl@usehooks{stringprocess}{}%
1899     \expandafter\bbl@stringdef
1900       \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.

```
1901 \ifx\bbl@opt@strings\relax
1902   \def\bbl@scset#1#2{\def#1{\bbl@encoded#2}}
1903   \bbl@patchuclc
1904   \let\bbl@encoded\relax
1905   \def\bbl@encoded@uclc#1{%
1906     \@inmathwarn#1%
1907     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
1908       \expandafter\ifx\csname ?\string#1\endcsname\relax
1909         \TextSymbolUnavailable#1%
1910       \else
1911         \csname ?\string#1\endcsname
1912       \fi
1913     \else
1914       \csname\cf@encoding\string#1\endcsname
1915     \fi}
1916 \else
1917   \def\bbl@scset#1#2{\def#1{#2}}
1918 \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.

```
1919 ⟨⟨*Macros local to BabelCommands⟩⟩ ≡
1920 \def\SetStringLoop##1##2{%
1921   \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
1922   \count@\z@
1923   \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
1924     \advance\count@\@ne
1925     \toks@\expandafter{\bbl@tempa}%
1926     \bbl@exp{%
1927       \\\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
1928       \count@=\the\count@\relax}}%
1929 ⟨⟨/Macros local to BabelCommands⟩⟩
```

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

```
1930 \def\bbl@aftercmds#1{%
1931   \toks@\expandafter{\bbl@scafter#1}%
1932   \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.

```
1933 ⟨⟨*Macros local to BabelCommands⟩⟩ ≡
1934   \newcommand\SetCase[3][]{%
1935     \bbl@patchuclc
1936     \bbl@forlang\bbl@tempa{%
1937       \bbl@carg\bbl@encstring{\bbl@tempa @bbl@uclc}{\bbl@tempa##1}%
1938       \bbl@carg\bbl@encstring{\bbl@tempa @bbl@uc}{##2}%
1939       \bbl@carg\bbl@encstring{\bbl@tempa @bbl@lc}{##3}}}%
1940 ⟨⟨/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.

```
1941 ⟨⟨*Macros local to BabelCommands⟩⟩ ≡
1942   \newcommand\SetHyphenMap[1]{%
1943     \bbl@forlang\bbl@tempa{%
1944       \expandafter\bbl@stringdef
1945         \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
1946 ⟨⟨/Macros local to BabelCommands⟩⟩
```

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

```
1947 \newcommand\BabelLower[2]{% one to one.
1948   \ifnum\lccode#1=#2\else
1949     \babel@savevariable{\lccode#1}%
1950     \lccode#1=#2\relax
1951   \fi}
1952 \newcommand\BabelLowerMM[4]{% many-to-many
1953   \@tempcnta=#1\relax
1954   \@tempcntb=#4\relax
1955   \def\bbl@tempa{%
1956     \ifnum\@tempcnta>#2\else
1957       \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
1958       \advance\@tempcnta#3\relax
1959       \advance\@tempcntb#3\relax
1960       \expandafter\bbl@tempa
1961     \fi}%
1962   \bbl@tempa}
1963 \newcommand\BabelLowerMO[4]{% many-to-one
1964   \@tempcnta=#1\relax
1965   \def\bbl@tempa{%
1966     \ifnum\@tempcnta>#2\else
1967       \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
1968       \advance\@tempcnta#3
1969       \expandafter\bbl@tempa
1970     \fi}%
1971   \bbl@tempa}
```

The following package options control the behavior of hyphenation mapping.

```
1972 ⟨⟨*More package options⟩⟩ ≡
1973 \DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\z@}
1974 \DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap\@ne}
1975 \DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\tw@}
1976 \DeclareOption{hyphenmap=other}{\chardef\bbl@opt@hyphenmap\thr@@}
1977 \DeclareOption{hyphenmap=other*}{\chardef\bbl@opt@hyphenmap4\relax}
1978 ⟨⟨/More package options⟩⟩
```

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

```
1979 \AtEndOfPackage{%
1980   \ifx\bbl@opt@hyphenmap\@undefined
1981     \bbl@xin@{,}{\bbl@language@opts}%
1982     \chardef\bbl@opt@hyphenmap\ifin@4\else\@ne\fi
1983   \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.

```
1984 \newcommand\setlocalecaption{%  TODO. Catch typos.
1985   \@ifstar\bbl@setcaption@s\bbl@setcaption@x}
1986 \def\bbl@setcaption@x#1#2#3{%  language caption-name string
1987   \bbl@trim@def\bbl@tempa{#2}%
1988   \bbl@xin@{.template}{\bbl@tempa}%
1989   \ifin@
1990     \bbl@ini@captions@template{#3}{#1}%
1991   \else
1992     \edef\bbl@tempd{%
1993       \expandafter\expandafter\expandafter
1994       \strip@prefix\expandafter\meaning\csname captions#1\endcsname}%
1995     \bbl@xin@
1996       {\expandafter\string\csname #2name\endcsname}%
1997       {\bbl@tempd}%
1998     \ifin@ % Renew caption
1999       \bbl@xin@{\string\bbl@scset}{\bbl@tempd}%
2000       \ifin@
2001         \bbl@exp{%
2002           \\\bbl@ifsamestring{\bbl@tempa}{\languagename}%
2003             {\\\bbl@scset\<#2name>\<#1#2name>}%
2004             {}}%
2005       \else % Old way converts to new way
2006         \bbl@ifunset{#1#2name}%
2007           {\bbl@exp{%
2008             \\\bbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}%
2009             \\\bbl@ifsamestring{\bbl@tempa}{\languagename}%
2010               {\def\<#2name>{\<#1#2name>}}%
2011               {}}}%
2012           {}%
2013       \fi
2014     \else
2015       \bbl@xin@{\string\bbl@scset}{\bbl@tempd}% New
2016       \ifin@ % New way
2017         \bbl@exp{%
2018           \\\bbl@add\<captions#1>{\\\bbl@scset\<#2name>\<#1#2name>}%
2019           \\\bbl@ifsamestring{\bbl@tempa}{\languagename}%
2020             {\\\bbl@scset\<#2name>\<#1#2name>}%
2021             {}}%
2022       \else  % Old way, but defined in the new way
2023         \bbl@exp{%
2024           \\\bbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}%
2025           \\\bbl@ifsamestring{\bbl@tempa}{\languagename}%
2026             {\def\<#2name>{\<#1#2name>}}%
2027             {}}%
2028       \fi%
2029     \fi
2030     \@namedef{#1#2name}{#3}%
2031     \toks@\expandafter{\bbl@captionslist}%
2032     \bbl@exp{\\\in@{\<#2name>}{\the\toks@}}%
2033     \ifin@\else
2034       \bbl@exp{\\\bbl@add\\\bbl@captionslist{\<#2name>}}%
2035       \bbl@toglobal\bbl@captionslist
2036     \fi
```

```
2037    \fi}
2038 % \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.

```
2039 \bbl@trace{Macros related to glyphs}
2040 \def\set@low@box#1{\setbox\tw@\hbox{,}\setbox\z@\hbox{#1}%
2041    \dimen\z@\ht\z@ \advance\dimen\z@ -\ht\tw@%
2042    \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.

```
2043 \def\save@sf@q#1{\leavevmode
2044    \begingroup
2045      \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
2046    \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.

```
2047 \ProvideTextCommand{\quotedblbase}{OT1}{%
2048    \save@sf@q{\set@low@box{\textquotedblright\/}%
2049      \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.

```
2050 \ProvideTextCommandDefault{\quotedblbase}{%
2051    \UseTextSymbol{OT1}{\quotedblbase}}
```

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

```
2052 \ProvideTextCommand{\quotesinglbase}{OT1}{%
2053    \save@sf@q{\set@low@box{\textquoteright\/}%
2054      \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.

```
2055 \ProvideTextCommandDefault{\quotesinglbase}{%
2056    \UseTextSymbol{OT1}{\quotesinglbase}}
```

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

```
2057 \ProvideTextCommand{\guillemetleft}{OT1}{%
2058    \ifmmode
2059      \ll
2060    \else
2061      \save@sf@q{\nobreak
2062        \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2063    \fi}
2064 \ProvideTextCommand{\guillemetright}{OT1}{%
2065    \ifmmode
2066      \gg
2067    \else
2068      \save@sf@q{\nobreak
2069        \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2070    \fi}
2071 \ProvideTextCommand{\guillemotleft}{OT1}{%
```

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

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

```
2085 \ProvideTextCommandDefault{\guillemetleft}{%
2086    \UseTextSymbol{OT1}{\guillemetleft}}
2087 \ProvideTextCommandDefault{\guillemetright}{%
2088    \UseTextSymbol{OT1}{\guillemetright}}
2089 \ProvideTextCommandDefault{\guillemotleft}{%
2090    \UseTextSymbol{OT1}{\guillemotleft}}
2091 \ProvideTextCommandDefault{\guillemotright}{%
2092    \UseTextSymbol{OT1}{\guillemotright}}
```

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

```
2093 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2094    \ifmmode
2095      <%
2096    \else
2097      \save@sf@q{\nobreak
2098        \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
2099    \fi}
2100 \ProvideTextCommand{\guilsinglright}{OT1}{%
2101    \ifmmode
2102      >%
2103    \else
2104      \save@sf@q{\nobreak
2105        \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
2106    \fi}
```

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

```
2107 \ProvideTextCommandDefault{\guilsinglleft}{%
2108    \UseTextSymbol{OT1}{\guilsinglleft}}
2109 \ProvideTextCommandDefault{\guilsinglright}{%
2110    \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.

```
2111 \DeclareTextCommand{\ij}{OT1}{%
2112    i\kern-0.02em\bbl@allowhyphens j}
2113 \DeclareTextCommand{\IJ}{OT1}{%
2114    I\kern-0.02em\bbl@allowhyphens J}
2115 \DeclareTextCommand{\ij}{T1}{\char188}
2116 \DeclareTextCommand{\IJ}{T1}{\char156}
```

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

```
2117 \ProvideTextCommandDefault{\ij}{%
2118    \UseTextSymbol{OT1}{\ij}}
2119 \ProvideTextCommandDefault{\IJ}{%
2120    \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).

```
2121 \def\crrtic@{\hrule height0.1ex width0.3em}
2122 \def\crttic@{\hrule height0.1ex width0.33em}
2123 \def\ddj@{%
2124   \setbox0\hbox{d}\dimen@=\ht0
2125   \advance\dimen@1ex
2126   \dimen@.45\dimen@
2127   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2128   \advance\dimen@ii.5ex
2129   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2130 \def\DDJ@{%
2131   \setbox0\hbox{D}\dimen@=.55\ht0
2132   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2133   \advance\dimen@ii.15ex %                correction for the dash position
2134   \advance\dimen@ii-.15\fontdimen7\font %     correction for cmtt font
2135   \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
2136   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
2137 %
2138 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2139 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}
```

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

```
2140 \ProvideTextCommandDefault{\dj}{%
2141   \UseTextSymbol{OT1}{\dj}}
2142 \ProvideTextCommandDefault{\DJ}{%
2143   \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.

```
2144 \DeclareTextCommand{\SS}{OT1}{SS}
2145 \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
```
2146 \ProvideTextCommandDefault{\glq}{%
2147   \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}
```

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

```
2148 \ProvideTextCommand{\grq}{T1}{%
2149   \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}
2150 \ProvideTextCommand{\grq}{TU}{%
2151   \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
2152 \ProvideTextCommand{\grq}{OT1}{%
2153   \save@sf@q{\kern-.0125em
2154     \textormath{\textquoteleft}{\mbox{\textquoteleft}}%
2155     \kern.07em\relax}}
2156 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}\grq}
```

\glqq The 'german' double quotes.
\grqq
```
2157 \ProvideTextCommandDefault{\glqq}{%
2158   \textormath{\quotedblbase}{\mbox{\quotedblbase}}}
```

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

```
2159 \ProvideTextCommand{\grqq}{T1}{%
2160   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
```

```
2161 \ProvideTextCommand{\grqq}{TU}{%
2162    \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2163 \ProvideTextCommand{\grqq}{OT1}{%
2164    \save@sf@q{\kern-.07em
2165       \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}%
2166       \kern.07em\relax}}
2167 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}
```

\flq The 'french' single guillemets.
\frq
```
2168 \ProvideTextCommandDefault{\flq}{%
2169    \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
2170 \ProvideTextCommandDefault{\frq}{%
2171    \textormath{\guilsinglright}{\mbox{\guilsinglright}}}
```

\flqq The 'french' double guillemets.
\frqq
```
2172 \ProvideTextCommandDefault{\flqq}{%
2173    \textormath{\guillemetleft}{\mbox{\guillemetleft}}}
2174 \ProvideTextCommandDefault{\frqq}{%
2175    \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).

```
2176 \def\umlauthigh{%
2177    \def\bbl@umlauta##1{\leavevmode\bgroup%
2178       \accent\csname\f@encoding dqpos\endcsname
2179       ##1\bbl@allowhyphens\egroup}%
2180    \let\bbl@umlaute\bbl@umlauta}
2181 \def\umlautlow{%
2182    \def\bbl@umlauta{\protect\lower@umlaut}}
2183 \def\umlautelow{%
2184    \def\bbl@umlaute{\protect\lower@umlaut}}
2185 \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.

```
2186 \expandafter\ifx\csname U@D\endcsname\relax
2187    \csname newdimen\endcsname\U@D
2188 \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.

```
2189 \def\lower@umlaut#1{%
2190    \leavevmode\bgroup
2191       \U@D 1ex%
2192       {\setbox\z@\hbox{%
2193          \char\csname\f@encoding dqpos\endcsname}%
2194          \dimen@ -.45ex\advance\dimen@\ht\z@
2195          \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2196       \accent\csname\f@encoding dqpos\endcsname
2197       \fontdimen5\font\U@D #1%
2198    \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).

```
2199 \AtBeginDocument{%
2200   \DeclareTextCompositeCommand{\"}{OT1}{a}{\bbl@umlauta{a}}%
2201   \DeclareTextCompositeCommand{\"}{OT1}{e}{\bbl@umlaute{e}}%
2202   \DeclareTextCompositeCommand{\"}{OT1}{i}{\bbl@umlaute{\i}}%
2203   \DeclareTextCompositeCommand{\"}{OT1}{\i}{\bbl@umlaute{\i}}%
2204   \DeclareTextCompositeCommand{\"}{OT1}{o}{\bbl@umlauta{o}}%
2205   \DeclareTextCompositeCommand{\"}{OT1}{u}{\bbl@umlauta{u}}%
2206   \DeclareTextCompositeCommand{\"}{OT1}{A}{\bbl@umlauta{A}}%
2207   \DeclareTextCompositeCommand{\"}{OT1}{E}{\bbl@umlaute{E}}%
2208   \DeclareTextCompositeCommand{\"}{OT1}{I}{\bbl@umlaute{I}}%
2209   \DeclareTextCompositeCommand{\"}{OT1}{O}{\bbl@umlauta{O}}%
2210   \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.

```
2211 \ifx\l@english\@undefined
2212   \chardef\l@english\z@
2213 \fi
2214 % The following is used to cancel rules in ini files (see Amharic).
2215 \ifx\l@unhyphenated\@undefined
2216   \newlanguage\l@unhyphenated
2217 \fi
```

7.13 Layout

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

```
2218 \bbl@trace{Bidi layout}
2219 \providecommand\IfBabelLayout[3]{#3}%
2220 \newcommand\BabelPatchSection[1]{%
2221   \@ifundefined{#1}{}{%
2222     \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
2223     \@namedef{#1}{%
2224       \@ifstar{\bbl@presec@s{#1}}%
2225               {\@dblarg{\bbl@presec@x{#1}}}}}}
2226 \def\bbl@presec@x#1[#2]#3{%
2227   \bbl@exp{%
2228     \\\select@language@x{\bbl@main@language}%
2229     \\\bbl@cs{sspre@#1}%
2230     \\\bbl@cs{ss@#1}%
2231       [\\\foreignlanguage{\languagename}{\unexpanded{#2}}]%
2232       {\\\foreignlanguage{\languagename}{\unexpanded{#3}}}%
2233     \\\select@language@x{\languagename}}}
2234 \def\bbl@presec@s#1#2{%
2235   \bbl@exp{%
2236     \\\select@language@x{\bbl@main@language}%
2237     \\\bbl@cs{sspre@#1}%
2238     \\\bbl@cs{ss@#1}*%
2239       {\\\foreignlanguage{\languagename}{\unexpanded{#2}}}%
2240     \\\select@language@x{\languagename}}}
2241 \IfBabelLayout{sectioning}%
2242   {\BabelPatchSection{part}%
2243    \BabelPatchSection{chapter}%
2244    \BabelPatchSection{section}%
2245    \BabelPatchSection{subsection}%
2246    \BabelPatchSection{subsubsection}%
2247    \BabelPatchSection{paragraph}%
```

```
2248     \BabelPatchSection{subparagraph}%
2249     \def\babel@toc#1{%
2250       \select@language@x{\bbl@main@language}}}}{}
2251 \IfBabelLayout{captions}%
2252   {\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.

```
2253 \bbl@trace{Input engine specific macros}
2254 \ifcase\bbl@engine
2255   \input txtbabel.def
2256 \or
2257   \input luababel.def
2258 \or
2259   \input xebabel.def
2260 \fi
2261 \providecommand\babelfont{%
2262   \bbl@error
2263     {This macro is available only in LuaLaTeX and XeLaTeX.}%
2264     {Consider switching to these engines.}}
2265 \providecommand\babelprehyphenation{%
2266   \bbl@error
2267     {This macro is available only in LuaLaTeX.}%
2268     {Consider switching to that engine.}}
2269 \ifx\babelposthyphenation\@undefined
2270   \let\babelposthyphenation\babelprehyphenation
2271   \let\babelpatterns\babelprehyphenation
2272   \let\babelcharproperty\babelprehyphenation
2273 \fi
```

7.15 Creating and modifying languages

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

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

```
2299        \bbl@csarg\ifx{KVP@##1}\@nnil\else
2300          \bbl@error
2301            {Unknown key '##1' in \string\babelprovide}%
2302            {See the manual for valid keys}%
2303          \fi
2304          \bbl@csarg\def{KVP@##1}{##2}%
2305        \fi}%
2306  \chardef\bbl@howloaded=% 0:none; 1:ldf without ini; 2:ini
2307    \bbl@ifunset{date#2}\z@{\bbl@ifunset{bbl@llevel@#2}\@ne\tw@}%
2308  % == init ==
2309  \ifx\bbl@screset\@undefined
2310    \bbl@ldfinit
2311  \fi
2312  % == date (as option) ==
2313  % \ifx\bbl@KVP@date\@nnil\else
2314  % \fi
2315  % ==
2316  \let\bbl@lbkflag\relax % \@empty = do setup linebreak, only in 3 cases:
2317  \ifcase\bbl@howloaded
2318    \let\bbl@lbkflag\@empty % new
2319  \else
2320    \ifx\bbl@KVP@hyphenrules\@nnil\else
2321      \let\bbl@lbkflag\@empty
2322    \fi
2323    \ifx\bbl@KVP@import\@nnil\else
2324      \let\bbl@lbkflag\@empty
2325    \fi
2326  \fi
2327  % == import, captions ==
2328  \ifx\bbl@KVP@import\@nnil\else
2329    \bbl@exp{\\\bbl@ifblank{\bbl@KVP@import}}%
2330      {\ifx\bbl@initoload\relax
2331        \begingroup
2332          \def\BabelBeforeIni##1##2{\gdef\bbl@KVP@import{##1}\endinput}%
2333          \bbl@input@texini{#2}%
2334        \endgroup
2335      \else
2336        \xdef\bbl@KVP@import{\bbl@initoload}%
2337      \fi}%
2338      {}%
2339    \let\bbl@KVP@date\@empty
2340  \fi
2341  \let\bbl@KVP@captions@@\bbl@KVP@captions % TODO. A dirty hack
2342  \ifx\bbl@KVP@captions\@nnil
2343    \let\bbl@KVP@captions\bbl@KVP@import
2344  \fi
2345  % ==
2346  \ifx\bbl@KVP@transforms\@nnil\else
2347    \bbl@replace\bbl@KVP@transforms{ }{,}%
2348  \fi
2349  % == Load ini ==
2350  \ifcase\bbl@howloaded
2351    \bbl@provide@new{#2}%
2352  \else
2353    \bbl@ifblank{#1}%
2354      {}%  With \bbl@load@basic below
2355      {\bbl@provide@renew{#2}}%
2356  \fi
2357  % Post tasks
2358  % ----------
2359  % == subsequent calls after the first provide for a locale ==
2360  \ifx\bbl@inidata\@empty\else
2361    \bbl@extend@ini{#2}%
```

```
2362     \fi
2363     % == ensure captions ==
2364     \ifx\bbl@KVP@captions\@nnil\else
2365       \bbl@ifunset{bbl@extracaps@#2}%
2366         {\bbl@exp{\\\babelensure[exclude=\\\today]{#2}}}%
2367         {\bbl@exp{\\\babelensure[exclude=\\\today,
2368                   include=\[bbl@extracaps@#2]]{#2}}}%
2369       \bbl@ifunset{bbl@ensure@\languagename}%
2370         {\bbl@exp{%
2371           \\\DeclareRobustCommand\<bbl@ensure@\languagename>[1]{%
2372             \\\foreignlanguage{\languagename}%
2373             {####1}}}}%
2374         {}%
2375       \bbl@exp{%
2376         \\\bbl@toglobal\<bbl@ensure@\languagename>%
2377         \\\bbl@toglobal\<bbl@ensure@\languagename\space>}%
2378     \fi
2379     % ==
2380     % At this point all parameters are defined if 'import'. Now we
2381     % execute some code depending on them. But what about if nothing was
2382     % imported? We just set the basic parameters, but still loading the
2383     % whole ini file.
2384     \bbl@load@basic{#2}%
2385     % == script, language ==
2386     % Override the values from ini or defines them
2387     \ifx\bbl@KVP@script\@nnil\else
2388       \bbl@csarg\edef{sname@#2}{\bbl@KVP@script}%
2389     \fi
2390     \ifx\bbl@KVP@language\@nnil\else
2391       \bbl@csarg\edef{lname@#2}{\bbl@KVP@language}%
2392     \fi
2393     \ifcase\bbl@engine\or
2394       \bbl@ifunset{bbl@chrng@\languagename}{}%
2395         {\directlua{
2396           Babel.set_chranges_b('\bbl@cl{sbcp}', '\bbl@cl{chrng}') }}%
2397     \fi
2398     % == onchar ==
2399     \ifx\bbl@KVP@onchar\@nnil\else
2400       \bbl@luahyphenate
2401       \bbl@exp{%
2402         \\\AddToHook{env/document/before}{{\\\select@language{#2}{}}}}%
2403       \directlua{
2404         if Babel.locale_mapped == nil then
2405           Babel.locale_mapped = true
2406           Babel.linebreaking.add_before(Babel.locale_map, 1)
2407           Babel.loc_to_scr = {}
2408           Babel.chr_to_loc = Babel.chr_to_loc or {}
2409         end
2410         Babel.locale_props[\the\localeid].letters = false
2411       }%
2412       \bbl@xin@{ letters }{ \bbl@KVP@onchar\space}%
2413       \ifin@
2414         \directlua{
2415           Babel.locale_props[\the\localeid].letters = true
2416         }%
2417       \fi
2418       \bbl@xin@{ ids }{ \bbl@KVP@onchar\space}%
2419       \ifin@
2420         \ifx\bbl@starthyphens\@undefined % Needed if no explicit selection
2421           \AddBabelHook{babel-onchar}{beforestart}{{\bbl@starthyphens}}%
2422         \fi
2423         \bbl@exp{\\\bbl@add\\\bbl@starthyphens
2424           {\\\bbl@patterns@lua{\languagename}}}%
```

```
2425        % TODO - error/warning if no script
2426        \directlua{
2427          if Babel.script_blocks['\bbl@cl{sbcp}'] then
2428            Babel.loc_to_scr[\the\localeid] =
2429              Babel.script_blocks['\bbl@cl{sbcp}']
2430            Babel.locale_props[\the\localeid].lc = \the\localeid\space
2431            Babel.locale_props[\the\localeid].lg = \the\@nameuse{l@\languagename}\space
2432          end
2433        }%
2434      \fi
2435      \bbl@xin@{ fonts }{ \bbl@KVP@onchar\space}%
2436      \ifin@
2437        \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
2438        \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
2439        \directlua{
2440          if Babel.script_blocks['\bbl@cl{sbcp}'] then
2441            Babel.loc_to_scr[\the\localeid] =
2442              Babel.script_blocks['\bbl@cl{sbcp}']
2443          end}%
2444        \ifx\bbl@mapselect\@undefined  % TODO. almost the same as mapfont
2445          \AtBeginDocument{%
2446            \bbl@patchfont{{\bbl@mapselect}}%
2447            {\selectfont}}%
2448          \def\bbl@mapselect{%
2449            \let\bbl@mapselect\relax
2450            \edef\bbl@prefontid{\fontid\font}}%
2451          \def\bbl@mapdir##1{%
2452            {\def\languagename{##1}%
2453             \let\bbl@ifrestoring\@firstoftwo % To avoid font warning
2454             \bbl@switchfont
2455             \ifnum\fontid\font>\z@ % A hack, for the pgf nullfont hack
2456               \directlua{
2457                 Babel.locale_props[\the\csname bbl@id@@##1\endcsname]%
2458                       ['/\bbl@prefontid'] = \fontid\font\space}%
2459             \fi}}%
2460        \fi
2461        \bbl@exp{\\\bbl@add\\\bbl@mapselect{\\\bbl@mapdir{\languagename}}}%
2462      \fi
2463      % TODO - catch non-valid values
2464    \fi
2465    % == mapfont ==
2466    % For bidi texts, to switch the font based on direction
2467    \ifx\bbl@KVP@mapfont\@nnil\else
2468      \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}{}%
2469        {\bbl@error{Option '\bbl@KVP@mapfont' unknown for\\%
2470                   mapfont. Use 'direction'.%
2471                  {See the manual for details.}}}%
2472      \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
2473      \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
2474      \ifx\bbl@mapselect\@undefined % TODO. See onchar.
2475        \AtBeginDocument{%
2476          \bbl@patchfont{{\bbl@mapselect}}%
2477          {\selectfont}}%
2478        \def\bbl@mapselect{%
2479          \let\bbl@mapselect\relax
2480          \edef\bbl@prefontid{\fontid\font}}%
2481        \def\bbl@mapdir##1{%
2482          {\def\languagename{##1}%
2483           \let\bbl@ifrestoring\@firstoftwo % avoid font warning
2484           \bbl@switchfont
2485           \directlua{Babel.fontmap
2486             [\the\csname bbl@wdir@##1\endcsname]%
2487             [\bbl@prefontid]=\fontid\font}}}%
```

```
2488      \fi
2489      \bbl@exp{\\\bbl@add\\\bbl@mapselect{\\\bbl@mapdir{\languagename}}}%
2490   \fi
2491   % == Line breaking: intraspace, intrapenalty ==
2492   % For CJK, East Asian, Southeast Asian, if interspace in ini
2493   \ifx\bbl@KVP@intraspace\@nnil\else % We can override the ini or set
2494      \bbl@csarg\edef{intsp@#2}{\bbl@KVP@intraspace}%
2495   \fi
2496   \bbl@provide@intraspace
2497   % == Line breaking: CJK quotes == TODO -> @extras
2498   \ifcase\bbl@engine\or
2499      \bbl@xin@{/c}{/\bbl@cl{lnbrk}}%
2500      \ifin@
2501        \bbl@ifunset{bbl@quote@\languagename}{}%
2502          {\directlua{
2503             Babel.locale_props[\the\localeid].cjk_quotes = {}
2504             local cs = 'op'
2505             for c in string.utfvalues(%
2506                [[\csname bbl@quote@\languagename\endcsname]]) do
2507               if Babel.cjk_characters[c].c == 'qu' then
2508                 Babel.locale_props[\the\localeid].cjk_quotes[c] = cs
2509               end
2510               cs = ( cs == 'op') and 'cl' or 'op'
2511             end
2512          }}%
2513      \fi
2514   \fi
2515   % == Line breaking: justification ==
2516   \ifx\bbl@KVP@justification\@nnil\else
2517      \let\bbl@KVP@linebreaking\bbl@KVP@justification
2518   \fi
2519   \ifx\bbl@KVP@linebreaking\@nnil\else
2520      \bbl@xin@{,\bbl@KVP@linebreaking,}%
2521        {,elongated,kashida,cjk,padding,unhyphenated,}%
2522      \ifin@
2523        \bbl@csarg\xdef
2524          {lnbrk@\languagename}{\expandafter\@car\bbl@KVP@linebreaking\@nil}%
2525      \fi
2526   \fi
2527   \bbl@xin@{/e}{/\bbl@cl{lnbrk}}%
2528   \ifin@\else\bbl@xin@{/k}{/\bbl@cl{lnbrk}}\fi
2529   \ifin@\bbl@arabicjust\fi
2530   \bbl@xin@{/p}{/\bbl@cl{lnbrk}}%
2531   \ifin@\AtBeginDocument{\@nameuse{bbl@tibetanjust}}\fi
2532   % == Line breaking: hyphenate.other.(locale|script) ==
2533   \ifx\bbl@lbkflag\@empty
2534      \bbl@ifunset{bbl@hyotl@\languagename}{}%
2535        {\bbl@csarg\bbl@replace{hyotl@\languagename}{ }{,}%
2536         \bbl@startcommands*{\languagename}{}%
2537           \bbl@csarg\bbl@foreach{hyotl@\languagename}{%
2538             \ifcase\bbl@engine
2539               \ifnum##1<257
2540                 \SetHyphenMap{\BabelLower{##1}{##1}}%
2541               \fi
2542             \else
2543               \SetHyphenMap{\BabelLower{##1}{##1}}%
2544             \fi}%
2545         \bbl@endcommands}%
2546      \bbl@ifunset{bbl@hyots@\languagename}{}%
2547        {\bbl@csarg\bbl@replace{hyots@\languagename}{ }{,}%
2548         \bbl@csarg\bbl@foreach{hyots@\languagename}{%
2549           \ifcase\bbl@engine
2550             \ifnum##1<257
```

```
2551            \global\lccode##1=##1\relax
2552          \fi
2553        \else
2554          \global\lccode##1=##1\relax
2555        \fi}}%
2556  \fi
2557  % == Counters: maparabic ==
2558  % Native digits, if provided in ini (TeX level, xe and lua)
2559  \ifcase\bbl@engine\else
2560    \bbl@ifunset{bbl@dgnat@\languagename}{}%
2561      {\expandafter\ifx\csname bbl@dgnat@\languagename\endcsname\@empty\else
2562        \expandafter\expandafter\expandafter
2563        \bbl@setdigits\csname bbl@dgnat@\languagename\endcsname
2564        \ifx\bbl@KVP@maparabic\@nnil\else
2565          \ifx\bbl@latinarabic\@undefined
2566            \expandafter\let\expandafter\@arabic
2567              \csname bbl@counter@\languagename\endcsname
2568          \else   % ie, if layout=counters, which redefines \@arabic
2569            \expandafter\let\expandafter\bbl@latinarabic
2570              \csname bbl@counter@\languagename\endcsname
2571          \fi
2572        \fi
2573      \fi}%
2574  \fi
2575  % == Counters: mapdigits ==
2576  % > luababel.def
2577  % == Counters: alph, Alph ==
2578  \ifx\bbl@KVP@alph\@nnil\else
2579    \bbl@exp{%
2580      \\\bbl@add\<bbl@preextras@\languagename>{%
2581        \\\babel@save\\\@alph
2582        \let\\\@alph\<bbl@cntr@\bbl@KVP@alph @\languagename>}}%
2583  \fi
2584  \ifx\bbl@KVP@Alph\@nnil\else
2585    \bbl@exp{%
2586      \\\bbl@add\<bbl@preextras@\languagename>{%
2587        \\\babel@save\\\@Alph
2588        \let\\\@Alph\<bbl@cntr@\bbl@KVP@Alph @\languagename>}}%
2589  \fi
2590  % == Calendars ==
2591  \ifx\bbl@KVP@calendar\@nnil
2592    \edef\bbl@KVP@calendar{\bbl@cl{calpr}}%
2593  \fi
2594  \def\bbl@tempe##1 ##2\@@{% Get first calendar
2595    \def\bbl@tempa{##1}}%
2596    \bbl@exp{\\\bbl@tempe\bbl@KVP@calendar\space\\\@@}%
2597  \def\bbl@tempe##1.##2.##3\@@{%
2598    \def\bbl@tempc{##1}%
2599    \def\bbl@tempb{##2}}%
2600  \expandafter\bbl@tempe\bbl@tempa..\@@
2601  \bbl@csarg\edef{calpr@\languagename}{%
2602    \ifx\bbl@tempc\@empty\else
2603      calendar=\bbl@tempc
2604    \fi
2605    \ifx\bbl@tempb\@empty\else
2606      ,variant=\bbl@tempb
2607    \fi}%
2608  % == engine specific extensions ==
2609  % Defined in XXXbabel.def
2610  \bbl@provide@extra{#2}%
2611  % == require.babel in ini ==
2612  % To load or reaload the babel-*.tex, if require.babel in ini
2613  \ifx\bbl@beforestart\relax\else  % But not in doc aux or body
```

```
2614    \bbl@ifunset{bbl@rqtex@\languagename}{}%
2615      {\expandafter\ifx\csname bbl@rqtex@\languagename\endcsname\@empty\else
2616        \let\BabelBeforeIni\@gobbletwo
2617        \chardef\atcatcode=\catcode`\@
2618        \catcode`\@=11\relax
2619        \bbl@input@texini{\bbl@cs{rqtex@\languagename}}%
2620        \catcode`\@=\atcatcode
2621        \let\atcatcode\relax
2622        \global\bbl@csarg\let{rqtex@\languagename}\relax
2623      \fi}%
2624    \bbl@foreach\bbl@calendars{%
2625      \bbl@ifunset{bbl@ca@##1}{%
2626        \chardef\atcatcode=\catcode`\@
2627        \catcode`\@=11\relax
2628        \InputIfFileExists{babel-ca-##1.tex}{}{}%
2629        \catcode`\@=\atcatcode
2630        \let\atcatcode\relax}%
2631      {}}%
2632  \fi
2633  % == frenchspacing ==
2634  \ifcase\bbl@howloaded\in@true\else\in@false\fi
2635  \ifin@\else\bbl@xin@{typography/frenchspacing}{\bbl@key@list}\fi
2636  \ifin@
2637    \bbl@extras@wrap{\\\bbl@pre@fs}%
2638      {\bbl@pre@fs}%
2639      {\bbl@post@fs}%
2640  \fi
2641  % == transforms ==
2642  % > luababel.def
2643  % == main ==
2644  \ifx\bbl@KVP@main\@nnil  % Restore only if not 'main'
2645    \let\languagename\bbl@savelangname
2646    \chardef\localeid\bbl@savelocaleid\relax
2647  \fi
2648  % == hyphenrules (apply if current) ==
2649  \ifx\bbl@KVP@hyphenrules\@nnil\else
2650    \ifnum\bbl@savelocaleid=\localeid
2651      \language\@nameuse{l@\languagename}%
2652    \fi
2653  \fi}
```

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

```
2654  \def\bbl@provide@new#1{%
2655    \@namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
2656    \@namedef{extras#1}{}%
2657    \@namedef{noextras#1}{}%
2658    \bbl@startcommands*{#1}{captions}%
2659      \ifx\bbl@KVP@captions\@nnil %       and also if import, implicit
2660        \def\bbl@tempb##1{%              elt for \bbl@captionslist
2661          \ifx##1\@empty\else
2662            \bbl@exp{%
2663              \\\SetString\\##1{%
2664                \\\bbl@nocaption{\bbl@stripslash##1}{#1\bbl@stripslash##1}}}%
2665            \expandafter\bbl@tempb
2666          \fi}%
2667        \expandafter\bbl@tempb\bbl@captionslist\@empty
2668      \else
2669        \ifx\bbl@initoload\relax
2670          \bbl@read@ini{\bbl@KVP@captions}2%  % Here letters cat = 11
2671        \else
2672          \bbl@read@ini{\bbl@initoload}2%      % Same
2673        \fi
```

```
2674      \fi
2675  \StartBabelCommands*{#1}{date}%
2676      \ifx\bbl@KVP@date\@nnil
2677        \bbl@exp{%
2678          \\\SetString\\\today{\\\bbl@nocaption{today}{#1today}}}%
2679      \else
2680        \bbl@savetoday
2681        \bbl@savedate
2682      \fi
2683  \bbl@endcommands
2684  \bbl@load@basic{#1}%
2685  % == hyphenmins == (only if new)
2686  \bbl@exp{%
2687    \gdef\<#1hyphenmins>{%
2688      {\bbl@ifunset{bbl@lfthm@#1}{2}{\bbl@cs{lfthm@#1}}}%
2689      {\bbl@ifunset{bbl@rgthm@#1}{3}{\bbl@cs{rgthm@#1}}}}}%
2690  % == hyphenrules (also in renew) ==
2691  \bbl@provide@hyphens{#1}%
2692  \ifx\bbl@KVP@main\@nnil\else
2693      \expandafter\main@language\expandafter{#1}%
2694  \fi}
2695 %
2696 \def\bbl@provide@renew#1{%
2697    \ifx\bbl@KVP@captions\@nnil\else
2698      \StartBabelCommands*{#1}{captions}%
2699        \bbl@read@ini{\bbl@KVP@captions}2%    % Here all letters cat = 11
2700      \EndBabelCommands
2701    \fi
2702    \ifx\bbl@KVP@date\@nnil\else
2703      \StartBabelCommands*{#1}{date}%
2704        \bbl@savetoday
2705        \bbl@savedate
2706      \EndBabelCommands
2707    \fi
2708  % == hyphenrules (also in new) ==
2709    \ifx\bbl@lbkflag\@empty
2710      \bbl@provide@hyphens{#1}%
2711    \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.)

```
2712 \def\bbl@load@basic#1{%
2713    \ifcase\bbl@howloaded\or\or
2714      \ifcase\csname bbl@llevel@\languagename\endcsname
2715        \bbl@csarg\let{lname@\languagename}\relax
2716      \fi
2717    \fi
2718    \bbl@ifunset{bbl@lname@#1}%
2719      {\def\BabelBeforeIni##1##2{%
2720        \begingroup
2721          \let\bbl@ini@captions@aux\@gobbletwo
2722          \def\bbl@inidate ####1.####2.####3.####4\relax ####5####6{}%
2723          \bbl@read@ini{##1}1%
2724          \ifx\bbl@initoload\relax\endinput\fi
2725        \endgroup}%
2726      \begingroup        % boxed, to avoid extra spaces:
2727        \ifx\bbl@initoload\relax
2728          \bbl@input@texini{#1}%
2729        \else
2730          \setbox\z@\hbox{\BabelBeforeIni{\bbl@initoload}{}}%
2731        \fi
2732      \endgroup}%
```

```
2733        {}}
```

The hyphenrules option is handled with an auxiliary macro. This macro is called in three cases: when a language is first declared with \babelprovide, with hyphenrules and with import.

```
2734 \def\bbl@provide@hyphens#1{%
2735   \@tempcnta\m@ne   % a flag
2736   \ifx\bbl@KVP@hyphenrules\@nnil\else
2737     \bbl@replace\bbl@KVP@hyphenrules{ }{,}%
2738     \bbl@foreach\bbl@KVP@hyphenrules{%
2739       \ifnum\@tempcnta=\m@ne   % if not yet found
2740         \bbl@ifsamestring{##1}{+}%
2741           {\bbl@carg\addlanguage{l@##1}}%
2742           {}%
2743         \bbl@ifunset{l@##1}% After a possible +
2744           {}%
2745           {\@tempcnta\@nameuse{l@##1}}%
2746       \fi}%
2747     \ifnum\@tempcnta=\m@ne
2748       \bbl@warning{%
2749         Requested 'hyphenrules' for '\languagename' not found:\\%
2750         \bbl@KVP@hyphenrules.\\%
2751         Using the default value. Reported}%
2752     \fi
2753   \fi
2754   \ifnum\@tempcnta=\m@ne           % if no opt or no language in opt found
2755     \ifx\bbl@KVP@captions@@\@nnil % TODO. Hackish. See above.
2756       \bbl@ifunset{bbl@hyphr@#1}{}% use value in ini, if exists
2757         {\bbl@exp{\\\bbl@ifblank{\bbl@cs{hyphr@#1}}}%
2758           {}%
2759           {\bbl@ifunset{l@\bbl@cl{hyphr}}%
2760             {}%                    if hyphenrules found:
2761             {\@tempcnta\@nameuse{l@\bbl@cl{hyphr}}}}}%
2762     \fi
2763   \fi
2764   \bbl@ifunset{l@#1}%
2765     {\ifnum\@tempcnta=\m@ne
2766       \bbl@carg\adddialect{l@#1}\language
2767     \else
2768       \bbl@carg\adddialect{l@#1}\@tempcnta
2769     \fi}%
2770     {\ifnum\@tempcnta=\m@ne\else
2771       \global\bbl@carg\chardef{l@#1}\@tempcnta
2772     \fi}}
```

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

```
2773 \def\bbl@input@texini#1{%
2774   \bbl@bsphack
2775     \bbl@exp{%
2776       \catcode`\\\%=14 \catcode`\\\\=0
2777       \catcode`\\\{=1  \catcode`\\\}=2
2778       \lowercase{\\\InputIfFileExists{babel-#1.tex}{}{}}%
2779       \catcode`\\\%=\the\catcode`\%\relax
2780       \catcode`\\\\=\the\catcode`\\\relax
2781       \catcode`\\\{=\the\catcode`\{\relax
2782       \catcode`\\\}=\the\catcode`\}\relax}%
2783   \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.

```
2784 \def\bbl@iniline#1\bbl@iniline{%
2785   \@ifnextchar[\bbl@inisect{\@ifnextchar;\bbl@iniskip\bbl@inistore}#1\@@}% ]
2786 \def\bbl@inisect[#1]#2\@@{\def\bbl@section{#1}}
2787 \def\bbl@iniskip#1\@@{}%      if starts with ;
```

```
2788 \def\bbl@inistore#1=#2\@@{%        full (default)
2789   \bbl@trim@def\bbl@tempa{#1}%
2790   \bbl@trim\toks@{#2}%
2791   \bbl@xin@{;\bbl@section/\bbl@tempa;}{\bbl@key@list}%
2792   \ifin@\else
2793     \bbl@xin@{,identification/include.}%
2794              {,\bbl@section/\bbl@tempa}%
2795     \ifin@\edef\bbl@required@inis{\the\toks@}\fi
2796     \bbl@exp{%
2797       \\\g@addto@macro\\\bbl@inidata{%
2798         \\\bbl@elt{\bbl@section}{\bbl@tempa}{\the\toks@}}}%
2799   \fi}
2800 \def\bbl@inistore@min#1=#2\@@{%  minimal (maybe set in \bbl@read@ini)
2801   \bbl@trim@def\bbl@tempa{#1}%
2802   \bbl@trim\toks@{#2}%
2803   \bbl@xin@{.identification.}{.\bbl@section.}%
2804   \ifin@
2805     \bbl@exp{\\\g@addto@macro\\\bbl@inidata{%
2806       \\\bbl@elt{identification}{\bbl@tempa}{\the\toks@}}}%
2807   \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.

```
2808 \def\bbl@loop@ini{%
2809   \loop
2810     \if T\ifeof\bbl@readstream F\fi T\relax % Trick, because inside \loop
2811       \endlinechar\m@ne
2812       \read\bbl@readstream to \bbl@line
2813       \endlinechar`\^^M
2814       \ifx\bbl@line\@empty\else
2815         \expandafter\bbl@iniline\bbl@line\bbl@iniline
2816       \fi
2817   \repeat}
2818 \ifx\bbl@readstream\@undefined
2819   \csname newread\endcsname\bbl@readstream
2820 \fi
2821 \def\bbl@read@ini#1#2{%
2822   \global\let\bbl@extend@ini\@gobble
2823   \openin\bbl@readstream=babel-#1.ini
2824   \ifeof\bbl@readstream
2825     \bbl@error
2826       {There is no ini file for the requested language\\%
2827        (#1: \languagename). Perhaps you misspelled it or your\\%
2828        installation is not complete.}%
2829       {Fix the name or reinstall babel.}%
2830   \else
2831     % == Store ini data in \bbl@inidata ==
2832     \catcode`\[=12 \catcode`\]=12 \catcode`\==12 \catcode`\&=12
2833     \catcode`\;=12 \catcode`\|=12 \catcode`\%=14 \catcode`\-=12
2834     \bbl@info{Importing
2835               \ifcase#2font and identification \or basic \fi
2836                data for \languagename\\%
2837              from babel-#1.ini. Reported}%
2838     \ifnum#2=\z@
2839       \global\let\bbl@inidata\@empty
2840       \let\bbl@inistore\bbl@inistore@min     % Remember it's local
2841     \fi
2842     \def\bbl@section{identification}%
2843     \let\bbl@required@inis\@empty
```

```
2844    \bbl@exp{\\\bbl@inistore tag.ini=#1\\\@@}%
2845    \bbl@inistore load.level=#2\@@
2846    \bbl@loop@ini
2847    \ifx\bbl@required@inis\@empty\else
2848      \bbl@replace\bbl@required@inis{ }{,}%
2849      \bbl@foreach\bbl@required@inis{%
2850        \openin\bbl@readstream=babel-##1.ini
2851        \bbl@loop@ini}%
2852    \fi
2853    % == Process stored data ==
2854    \bbl@csarg\xdef{lini@\languagename}{#1}%
2855    \bbl@read@ini@aux
2856    % == 'Export' data ==
2857    \bbl@ini@exports{#2}%
2858    \global\bbl@csarg\let{inidata@\languagename}\bbl@inidata
2859    \global\let\bbl@inidata\@empty
2860    \bbl@exp{\\\bbl@add@list\\\bbl@ini@loaded{\languagename}}%
2861    \bbl@toglobal\bbl@ini@loaded
2862  \fi
2863  \closein\bbl@readstream}
2864 \def\bbl@read@ini@aux{%
2865  \let\bbl@savestrings\@empty
2866  \let\bbl@savetoday\@empty
2867  \let\bbl@savedate\@empty
2868  \def\bbl@elt##1##2##3{%
2869    \def\bbl@section{##1}%
2870    \in@{=date.}{=##1}% Find a better place
2871    \ifin@
2872      \bbl@ifunset{bbl@inikv@##1}%
2873        {\bbl@ini@calendar{##1}}%
2874        {}%
2875    \fi
2876    \bbl@ifunset{bbl@inikv@##1}{}%
2877      {\csname bbl@inikv@##1\endcsname{##2}{##3}}}%
2878  \bbl@inidata}
```

A variant to be used when the ini file has been already loaded, because it's not the first \babelprovide for this language.

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

```
2904     \bbl@savedate
2905   \bbl@endcommands}
```

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

```
2906 \def\bbl@ini@calendar#1{%
2907   \lowercase{\def\bbl@tempa{=#1=}}%
2908   \bbl@replace\bbl@tempa{=date.gregorian}{}%
2909   \bbl@replace\bbl@tempa{=date.}{}%
2910   \in@{.licr=}{#1=}%
2911   \ifin@
2912     \ifcase\bbl@engine
2913       \bbl@replace\bbl@tempa{.licr=}{}%
2914     \else
2915       \let\bbl@tempa\relax
2916     \fi
2917   \fi
2918   \ifx\bbl@tempa\relax\else
2919     \bbl@replace\bbl@tempa{=}{}%
2920     \ifx\bbl@tempa\@empty\else
2921       \xdef\bbl@calendars{\bbl@calendars,\bbl@tempa}%
2922     \fi
2923     \bbl@exp{%
2924       \def\<bbl@inikv@#1>####1####2{%
2925         \\\bbl@inidate####1...\relax{####2}{\bbl@tempa}}}%
2926 \fi}
```

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

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

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

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

Key-value pairs are treated differently depending on the section in the ini file. The following macros are the readers for identification and typography. Note \bbl@ini@exports is called always (via \bbl@inisec), while \bbl@after@ini must be called explicitly after \bbl@read@ini if necessary. Although BCP 47 doesn't treat '-x-' as an extension, the CLDR and many other sources do (as a *private use extension*). For consistency with other single-letter subtags or 'singletons', here is considered an extension, too.

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

```
2951 \def\bbl@ini@exports#1{%
2952   % Identification always exported
2953   \bbl@iniwarning{}%
2954   \ifcase\bbl@engine
2955     \bbl@iniwarning{.pdflatex}%
2956   \or
2957     \bbl@iniwarning{.lualatex}%
2958   \or
2959     \bbl@iniwarning{.xelatex}%
2960   \fi%
2961   \bbl@exportkey{llevel}{identification.load.level}{}%
2962   \bbl@exportkey{elname}{identification.name.english}{}%
2963   \bbl@exp{\\\bbl@exportkey{lname}{identification.name.opentype}%
2964     {\csname bbl@elname@\languagename\endcsname}}%
2965   \bbl@exportkey{tbcp}{identification.tag.bcp47}{}%
2966   \bbl@exportkey{lbcp}{identification.language.tag.bcp47}{}%
2967   \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
2968   \bbl@exportkey{esname}{identification.script.name}{}%
2969   \bbl@exp{\\\bbl@exportkey{sname}{identification.script.name.opentype}%
2970     {\csname bbl@esname@\languagename\endcsname}}%
2971   \bbl@exportkey{sbcp}{identification.script.tag.bcp47}{}%
2972   \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
2973   \bbl@exportkey{rbcp}{identification.region.tag.bcp47}{}%
2974   \bbl@exportkey{vbcp}{identification.variant.tag.bcp47}{}%
2975   \bbl@exportkey{extt}{identification.extension.t.tag.bcp47}{}%
2976   \bbl@exportkey{extu}{identification.extension.u.tag.bcp47}{}%
2977   \bbl@exportkey{extx}{identification.extension.x.tag.bcp47}{}%
2978   % Also maps bcp47 -> languagename
2979   \ifbbl@bcptoname
2980     \bbl@csarg\xdef{bcp@map@\bbl@cl{tbcp}}{\languagename}%
2981   \fi
2982   % Conditional
2983   \ifnum#1>\z@          % 0 = only info, 1, 2 = basic, (re)new
2984     \bbl@exportkey{calpr}{date.calendar.preferred}{}%
2985     \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
2986     \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
2987     \bbl@exportkey{lfthm}{typography.lefthyphenmin}{2}%
2988     \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
2989     \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
2990     \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
2991     \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
2992     \bbl@exportkey{intsp}{typography.intraspace}{}%
2993     \bbl@exportkey{frspc}{typography.frenchspacing}{u}%
2994     \bbl@exportkey{chrng}{characters.ranges}{}%
2995     \bbl@exportkey{quote}{characters.delimiters.quotes}{}%
2996     \bbl@exportkey{dgnat}{numbers.digits.native}{}%
2997     \ifnum#1=\tw@          % only (re)new
2998       \bbl@exportkey{rqtex}{identification.require.babel}{}%
2999       \bbl@toglobal\bbl@savetoday
3000       \bbl@toglobal\bbl@savedate
3001       \bbl@savestrings
3002     \fi
3003   \fi}
```

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

```
3004 \def\bbl@inikv#1#2{%        key=value
3005   \toks@{#2}%               This hides #'s from ini values
3006   \bbl@csarg\edef{@kv@\bbl@section.#1}{\the\toks@}}
```

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

```
3007 \let\bbl@inikv@identification\bbl@inikv
3008 \let\bbl@inikv@date\bbl@inikv
3009 \let\bbl@inikv@typography\bbl@inikv
3010 \let\bbl@inikv@characters\bbl@inikv
```

125

```
3011 \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'.

```
3012 \def\bbl@inikv@counters#1#2{%
3013   \bbl@ifsamestring{#1}{digits}%
3014     {\bbl@error{The counter name 'digits' is reserved for mapping\\%
3015                 decimal digits}%
3016                {Use another name.}}%
3017     {}%
3018   \def\bbl@tempc{#1}%
3019   \bbl@trim@def{\bbl@tempb*}{#2}%
3020   \in@{.1$}{#1$}%
3021   \ifin@
3022     \bbl@replace\bbl@tempc{.1}{}%
3023     \bbl@csarg\protected@xdef{cntr@\bbl@tempc @\languagename}{%
3024       \noexpand\bbl@alphnumeral{\bbl@tempc}}%
3025   \fi
3026   \in@{.F.}{#1}%
3027   \ifin@\else\in@{.S.}{#1}\fi
3028   \ifin@
3029     \bbl@csarg\protected@xdef{cntr@#1@\languagename}{\bbl@tempb*}%
3030   \else
3031     \toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa
3032     \expandafter\bbl@buildifcase\bbl@tempb* \\ % Space after \\
3033     \bbl@csarg{\global\expandafter\let}{cntr@#1@\languagename}\bbl@tempa
3034   \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.

```
3035 \ifcase\bbl@engine
3036   \bbl@csarg\def{inikv@captions.licr}#1#2{%
3037     \bbl@ini@captions@aux{#1}{#2}}
3038 \else
3039   \def\bbl@inikv@captions#1#2{%
3040     \bbl@ini@captions@aux{#1}{#2}}
3041 \fi
```

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

```
3042 \def\bbl@ini@captions@template#1#2{% string language tempa=capt-name
3043   \bbl@replace\bbl@tempa{.template}{}%
3044   \def\bbl@toreplace{#1{}}%
3045   \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
3046   \bbl@replace\bbl@toreplace{[[}{\csname}%
3047   \bbl@replace\bbl@toreplace{[}{\csname the}%
3048   \bbl@replace\bbl@toreplace{]]}{name\endcsname{}}%
3049   \bbl@replace\bbl@toreplace{]}{\endcsname{}}%
3050   \bbl@xin@{,\bbl@tempa,}{,chapter,appendix,part,}%
3051   \ifin@
3052     \@nameuse{bbl@patch\bbl@tempa}%
3053     \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
3054   \fi
3055   \bbl@xin@{,\bbl@tempa,}{,figure,table,}%
3056   \ifin@
3057     \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
3058     \bbl@exp{\gdef\<fnum@\bbl@tempa>{%
3059       \\\bbl@ifunset{bbl@\bbl@tempa fmt@\\\languagename}%
3060         {\[fnum@\bbl@tempa]}%
3061         {\\\@nameuse{bbl@\bbl@tempa fmt@\\\languagename}}}}%
3062   \fi}
3063 \def\bbl@ini@captions@aux#1#2{%
3064   \bbl@trim@def\bbl@tempa{#1}%
```

```
3065    \bbl@xin@{.template}{\bbl@tempa}%
3066    \ifin@
3067      \bbl@ini@captions@template{#2}\languagename
3068    \else
3069      \bbl@ifblank{#2}%
3070        {\bbl@exp{%
3071          \toks@{\\\bbl@nocaption{\bbl@tempa}{\languagename\bbl@tempa name}}}}%
3072        {\bbl@trim\toks@{#2}}%
3073      \bbl@exp{%
3074        \\\bbl@add\\\bbl@savestrings{%
3075          \\\SetString\<\bbl@tempa name>{\the\toks@}}}%
3076      \toks@\expandafter{\bbl@captionslist}%
3077      \bbl@exp{\\\in@{\<\bbl@tempa name>}{\the\toks@}}%
3078      \ifin@\else
3079        \bbl@exp{%
3080          \\\bbl@add\<bbl@extracaps@\languagename>{\<\bbl@tempa name>}%
3081          \\\bbl@toglobal\<bbl@extracaps@\languagename>}%
3082      \fi
3083    \fi}
```

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

```
3084 \def\bbl@list@the{%
3085   part,chapter,section,subsection,subsubsection,paragraph,%
3086   subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,%
3087   table,page,footnote,mpfootnote,mpfn}
3088 \def\bbl@map@cnt#1{%  #1:roman,etc, // #2:enumi,etc
3089   \bbl@ifunset{bbl@map@#1@\languagename}%
3090     {\@nameuse{#1}}%
3091     {\@nameuse{bbl@map@#1@\languagename}}}
3092 \def\bbl@inikv@labels#1#2{%
3093   \in@{.map}{#1}%
3094   \ifin@
3095     \ifx\bbl@KVP@labels\@nnil\else
3096       \bbl@xin@{ map }{ \bbl@KVP@labels\space}%
3097       \ifin@
3098         \def\bbl@tempc{#1}%
3099         \bbl@replace\bbl@tempc{.map}{}%
3100         \in@{,#2,}{,arabic,roman,Roman,alph,Alph,fnsymbol,}%
3101         \bbl@exp{%
3102           \gdef\<bbl@map@\bbl@tempc @\languagename>%
3103             {\ifin@\<#2>\else\\\localcounter{#2}\fi}}%
3104         \bbl@foreach\bbl@list@the{%
3105           \bbl@ifunset{the##1}{}%
3106             {\bbl@exp{\let\\\bbl@tempd\<the##1>}%
3107              \bbl@exp{%
3108                \\\bbl@sreplace\<the##1>%
3109                  {\<\bbl@tempc>{##1}}{\\\bbl@map@cnt{\bbl@tempc}{##1}}%
3110                \\\bbl@sreplace\<the##1>%
3111                  {\<\@empty @\bbl@tempc>\<c@##1>}{\\\bbl@map@cnt{\bbl@tempc}{##1}}}%
3112              \expandafter\ifx\csname the##1\endcsname\bbl@tempd\else
3113                \toks@\expandafter\expandafter\expandafter{%
3114                  \csname the##1\endcsname}%
3115                \expandafter\xdef\csname the##1\endcsname{{\the\toks@}}%
3116              \fi}}%
3117       \fi
3118     \fi
3119 %
3120   \else
3121     %
3122     % The following code is still under study. You can test it and make
3123     % suggestions. Eg, enumerate.2 = ([enumi]).([enumii]). It's
3124     % language dependent.
3125     \in@{enumerate.}{#1}%
```

```
3126    \ifin@
3127      \def\bbl@tempa{#1}%
3128      \bbl@replace\bbl@tempa{enumerate.}{}%
3129      \def\bbl@toreplace{#2}%
3130      \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
3131      \bbl@replace\bbl@toreplace{[}{\csname the}%
3132      \bbl@replace\bbl@toreplace{]}{\endcsname{}}%
3133      \toks@\expandafter{\bbl@toreplace}%
3134      % TODO. Execute only once:
3135      \bbl@exp{%
3136        \\\bbl@add\<extras\languagename>{%
3137          \\\babel@save\<labelenum\romannumeral\bbl@tempa>%
3138          \def\<labelenum\romannumeral\bbl@tempa>{\the\toks@}}%
3139        \\\bbl@toglobal\<extras\languagename>}%
3140    \fi
3141  \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.

```
3142 \def\bbl@chaptype{chapter}
3143 \ifx\@makechapterhead\@undefined
3144   \let\bbl@patchchapter\relax
3145 \else\ifx\thechapter\@undefined
3146   \let\bbl@patchchapter\relax
3147 \else\ifx\ps@headings\@undefined
3148   \let\bbl@patchchapter\relax
3149 \else
3150   \def\bbl@patchchapter{%
3151     \global\let\bbl@patchchapter\relax
3152     \gdef\bbl@chfmt{%
3153       \bbl@ifunset{bbl@\bbl@chaptype fmt@\languagename}%
3154         {\@chapapp\space\thechapter}
3155         {\@nameuse{bbl@\bbl@chaptype fmt@\languagename}}}
3156     \bbl@add\appendix{\def\bbl@chaptype{appendix}}% Not harmful, I hope
3157     \bbl@sreplace\ps@headings{\@chapapp\ \thechapter}{\bbl@chfmt}%
3158     \bbl@sreplace\chaptermark{\@chapapp\ \thechapter}{\bbl@chfmt}%
3159     \bbl@sreplace\@makechapterhead{\@chapapp\space\thechapter}{\bbl@chfmt}%
3160     \bbl@toglobal\appendix
3161     \bbl@toglobal\ps@headings
3162     \bbl@toglobal\chaptermark
3163     \bbl@toglobal\@makechapterhead}
3164   \let\bbl@patchappendix\bbl@patchchapter
3165 \fi\fi\fi
3166 \ifx\@part\@undefined
3167   \let\bbl@patchpart\relax
3168 \else
3169   \def\bbl@patchpart{%
3170     \global\let\bbl@patchpart\relax
3171     \gdef\bbl@partformat{%
3172       \bbl@ifunset{bbl@partfmt@\languagename}%
3173         {\partname\nobreakspace\thepart}
3174         {\@nameuse{bbl@partfmt@\languagename}}}
3175     \bbl@sreplace\@part{\partname\nobreakspace\thepart}{\bbl@partformat}%
3176     \bbl@toglobal\@part}
3177 \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

```
3178 \let\bbl@calendar\@empty
3179 \DeclareRobustCommand\localedate[1][]{\bbl@localedate{#1}}
3180 \def\bbl@localedate#1#2#3#4{%
3181   \begingroup
```

```
3182      \edef\bbl@they{#2}%
3183      \edef\bbl@them{#3}%
3184      \edef\bbl@thed{#4}%
3185      \edef\bbl@tempe{%
3186        \bbl@ifunset{bbl@calpr@\languagename}{}{\bbl@cl{calpr}},%
3187        #1}%
3188      \bbl@replace\bbl@tempe{ }{}%
3189      \bbl@replace\bbl@tempe{CONVERT}{convert=}% Hackish
3190      \bbl@replace\bbl@tempe{convert}{convert=}%
3191      \let\bbl@ld@calendar\@empty
3192      \let\bbl@ld@variant\@empty
3193      \let\bbl@ld@convert\relax
3194      \def\bbl@tempb##1=##2\@@{\@namedef{bbl@ld@##1}{##2}}%
3195      \bbl@foreach\bbl@tempe{\bbl@tempb##1\@@}%
3196      \bbl@replace\bbl@ld@calendar{gregorian}{}%
3197      \ifx\bbl@ld@calendar\@empty\else
3198        \ifx\bbl@ld@convert\relax\else
3199          \babelcalendar[\bbl@they-\bbl@them-\bbl@thed]%
3200             {\bbl@ld@calendar}\bbl@they\bbl@them\bbl@thed
3201        \fi
3202      \fi
3203      \@nameuse{bbl@precalendar}% Remove, eg, +, -civil (-ca-islamic)
3204      \edef\bbl@calendar{% Used in \month..., too
3205        \bbl@ld@calendar
3206        \ifx\bbl@ld@variant\@empty\else
3207          .\bbl@ld@variant
3208        \fi}%
3209      \bbl@cased
3210        {\@nameuse{bbl@date@\languagename @\bbl@calendar}%
3211           \bbl@they\bbl@them\bbl@thed}%
3212   \endgroup}
3213 % eg: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
3214 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
3215    \bbl@trim@def\bbl@tempa{#1.#2}%
3216    \bbl@ifsamestring{\bbl@tempa}{months.wide}%      to savedate
3217      {\bbl@trim@def\bbl@tempa{#3}%
3218       \bbl@trim\toks@{#5}%
3219       \@temptokena\expandafter{\bbl@savedate}%
3220       \bbl@exp{%   Reverse order - in ini last wins
3221         \def\\\bbl@savedate{%
3222           \\\SetString\<month\romannumeral\bbl@tempa#6name>{\the\toks@}%
3223           \the\@temptokena}}}%
3224      {\bbl@ifsamestring{\bbl@tempa}{date.long}%     defined now
3225        {\lowercase{\def\bbl@tempb{#6}}%
3226         \bbl@trim@def\bbl@toreplace{#5}%
3227         \bbl@TG@@date
3228         \global\bbl@csarg\let{date@\languagename @\bbl@tempb}\bbl@toreplace
3229         \ifx\bbl@savetoday\@empty
3230           \bbl@exp{% TODO. Move to a better place.
3231             \\\AfterBabelCommands{%
3232               \def\<\languagename date>{\\\protect\<\languagename date >}%
3233               \\\newcommand\<\languagename date >[4][]{%
3234                 \\\bbl@usedategrouptrue
3235                 \<bbl@ensure@\languagename>{%
3236                   \\\localedate[####1]{####2}{####3}{####4}}}}%
3237             \def\\\bbl@savetoday{%
3238               \\\SetString\\\today{%
3239                 \<\languagename date>[convert]%
3240                   {\\\the\year}{\\\the\month}{\\\the\day}}}}%
3241         \fi}%
3242      {}}}
```

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).

```
3243 \let\bbl@calendar\@empty
3244 \newcommand\babelcalendar[2][\the\year-\the\month-\the\day]{%
3245   \@nameuse{bbl@ca@#2}#1\@@}
3246 \newcommand\BabelDateSpace{\nobreakspace}
3247 \newcommand\BabelDateDot{.\@}  % TODO. \let instead of repeating
3248 \newcommand\BabelDated[1]{{\number#1}}
3249 \newcommand\BabelDatedd[1]{{\ifnum#1<10 0\fi\number#1}}
3250 \newcommand\BabelDateM[1]{{\number#1}}
3251 \newcommand\BabelDateMM[1]{{\ifnum#1<10 0\fi\number#1}}
3252 \newcommand\BabelDateMMMM[1]{{%
3253   \csname month\romannumeral#1\bbl@calendar name\endcsname}}%
3254 \newcommand\BabelDatey[1]{{\number#1}}%
3255 \newcommand\BabelDateyy[1]{{%
3256   \ifnum#1<10 0\number#1 %
3257   \else\ifnum#1<100 \number#1 %
3258   \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3259   \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
3260   \else
3261     \bbl@error
3262       {Currently two-digit years are restricted to the\\
3263        range 0-9999.}%
3264       {There is little you can do. Sorry.}%
3265   \fi\fi\fi\fi}}
3266 \newcommand\BabelDateyyyy[1]{{\number#1}} % TODO - add leading 0
3267 \def\bbl@replace@finish@iii#1{%
3268   \bbl@exp{\def\\#1####1####2####3{\the\toks@}}}
3269 \def\bbl@TG@@date{%
3270   \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace{}}%
3271   \bbl@replace\bbl@toreplace{[.]}{\BabelDateDot{}}%
3272   \bbl@replace\bbl@toreplace{[d]}{\BabelDated{####3}}%
3273   \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{####3}}%
3274   \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{####2}}%
3275   \bbl@replace\bbl@toreplace{[MM]}{\BabelDateMM{####2}}%
3276   \bbl@replace\bbl@toreplace{[MMMM]}{\BabelDateMMMM{####2}}%
3277   \bbl@replace\bbl@toreplace{[y]}{\BabelDatey{####1}}%
3278   \bbl@replace\bbl@toreplace{[yy]}{\BabelDateyy{####1}}%
3279   \bbl@replace\bbl@toreplace{[yyyy]}{\BabelDateyyyy{####1}}%
3280   \bbl@replace\bbl@toreplace{[y|}{\bbl@datecntr[####1|}%
3281   \bbl@replace\bbl@toreplace{[m|}{\bbl@datecntr[####2|}%
3282   \bbl@replace\bbl@toreplace{[d|}{\bbl@datecntr[####3|}%
3283   \bbl@replace@finish@iii\bbl@toreplace}
3284 \def\bbl@datecntr{\expandafter\bbl@xdatecntr\expandafter}
3285 \def\bbl@xdatecntr#1|#2]{\localenumeral{#2}{#1}}
```

Transforms.

```
3286 \let\bbl@release@transforms\@empty
3287 \bbl@csarg\let{inikv@transforms.prehyphenation}\bbl@inikv
3288 \bbl@csarg\let{inikv@transforms.posthyphenation}\bbl@inikv
3289 \def\bbl@transforms@aux#1#2#3#4,#5\relax{%
3290   #1[#2]{#3}{#4}{#5}}
3291 \begingroup %  A hack. TODO. Don't require an specific order
3292   \catcode`\%=12
3293   \catcode`\&=14
3294   \gdef\bbl@transforms#1#2#3{&%
3295     \directlua{
3296       local str = [==[#2]==]
3297       str = str:gsub('%.%d+%.%d+$', '')
3298       token.set_macro('babeltempa', str)
3299     }&%
3300     \def\babeltempc{}&%
```

130

```
3301        \bbl@xin@{,\babeltempa,}{,\bbl@KVP@transforms,}&%
3302        \ifin@\else
3303          \bbl@xin@{:\babeltempa,}{,\bbl@KVP@transforms,}&%
3304        \fi
3305        \ifin@
3306          \bbl@foreach\bbl@KVP@transforms{&%
3307            \bbl@xin@{:\babeltempa,}{,##1,}&%
3308            \ifin@   &% font:font:transform syntax
3309              \directlua{
3310                local t = {}
3311                for m in string.gmatch('##1'..':', '(.-):') do
3312                  table.insert(t, m)
3313                end
3314                table.remove(t)
3315                token.set_macro('babeltempc', ',fonts=' .. table.concat(t, ' '))
3316              }&%
3317            \fi}&%
3318          \in@{.0$}{#2$}&%
3319          \ifin@
3320            \directlua{&% (\attribute) syntax
3321              local str = string.match([[\bbl@KVP@transforms]],
3322                            '%(([^%(]-)%)[^%)]-\babeltempa')
3323              if str == nil then
3324                token.set_macro('babeltempb', '')
3325              else
3326                token.set_macro('babeltempb', ',attribute=' .. str)
3327              end
3328            }&%
3329            \toks@{#3}&%
3330            \bbl@exp{&%
3331              \\\g@addto@macro\\\bbl@release@transforms{&%
3332                \relax   &% Closes previous \bbl@transforms@aux
3333                \\\bbl@transforms@aux
3334                  \\#1{label=\babeltempa\babeltempb\babeltempc}&%
3335                    {\languagename}{\the\toks@}}}&%
3336          \else
3337            \g@addto@macro\bbl@release@transforms{, {#3}}&%
3338          \fi
3339      \fi}
3340 \endgroup
```

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

```
3341 \def\bbl@provide@lsys#1{%
3342   \bbl@ifunset{bbl@lname@#1}%
3343     {\bbl@load@info{#1}}%
3344     {}%
3345   \bbl@csarg\let{lsys@#1}\@empty
3346   \bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}{}%
3347   \bbl@ifunset{bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{}%
3348   \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
3349   \bbl@ifunset{bbl@lname@#1}{}%
3350     {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
3351   \ifcase\bbl@engine\or\or
3352     \bbl@ifunset{bbl@prehc@#1}{}%
3353       {\bbl@exp{\\\bbl@ifblank{\bbl@cs{prehc@#1}}}%
3354         {}%
3355         {\ifx\bbl@xenohyph\@undefined
3356            \global\let\bbl@xenohyph\bbl@xenohyph@d
3357            \ifx\AtBeginDocument\@notprerr
3358              \expandafter\@secondoftwo   % to execute right now
3359            \fi
3360            \AtBeginDocument{%
```

```
3361            \bbl@patchfont{\bbl@xenohyph}%
3362            \expandafter\select@language\expandafter{\languagename}}%
3363        \fi}}%
3364    \fi
3365    \bbl@csarg\bbl@toglobal{lsys@#1}}
3366 \def\bbl@xenohyph@d{%
3367    \bbl@ifset{bbl@prehc@\languagename}%
3368       {\ifnum\hyphenchar\font=\defaulthyphenchar
3369          \iffontchar\font\bbl@cl{prehc}\relax
3370            \hyphenchar\font\bbl@cl{prehc}\relax
3371          \else\iffontchar\font"200B
3372            \hyphenchar\font"200B
3373          \else
3374            \bbl@warning
3375              {Neither 0 nor ZERO WIDTH SPACE are available\\%
3376               in the current font, and therefore the hyphen\\%
3377               will be printed. Try changing the fontspec's\\%
3378               'HyphenChar' to another value, but be aware\\%
3379               this setting is not safe (see the manual).\\%
3380               Reported}%
3381            \hyphenchar\font\defaulthyphenchar
3382          \fi\fi
3383        \fi}%
3384       {\hyphenchar\font\defaulthyphenchar}}
3385   % \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).

```
3386 \def\bbl@load@info#1{%
3387    \def\BabelBeforeIni##1##2{%
3388       \begingroup
3389       \bbl@read@ini{##1}0%
3390       \endinput          % babel- .tex may contain onlypreamble's
3391       \endgroup}%               boxed, to avoid extra spaces:
3392    {\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.

```
3393 \def\bbl@setdigits#1#2#3#4#5{%
3394    \bbl@exp{%
3395       \def\<\languagename digits>####1{%          ie, \langdigits
3396          \<bbl@digits@\languagename>####1\\\@nil}%
3397       \let\<bbl@cntr@digits@\languagename>\<\languagename digits>%
3398       \def\<\languagename counter>####1{%          ie, \langcounter
3399          \\\expandafter\<bbl@counter@\languagename>%
3400          \\\csname c@####1\endcsname}%
3401       \def\<bbl@counter@\languagename>####1{% ie, \bbl@counter@lang
3402          \\\expandafter\<bbl@digits@\languagename>%
3403          \\\number####1\\\@nil}}%
3404    \def\bbl@tempa##1##2##3##4##5{%
3405       \bbl@exp{%    Wow, quite a lot of hashes! :-(
3406          \def\<bbl@digits@\languagename>########1{%
3407             \\\ifx########1\\\@nil               % ie, \bbl@digits@lang
3408             \\\else
3409               \\\ifx0########1#1%
3410               \\\else\\\ifx1########1#2%
3411               \\\else\\\ifx2########1#3%
3412               \\\else\\\ifx3########1#4%
3413               \\\else\\\ifx4########1#5%
3414               \\\else\\\ifx5########1##1%
3415               \\\else\\\ifx6########1##2%
```

```
3416        \\\else\\\ifx7########1##3%
3417        \\\else\\\ifx8########1##4%
3418        \\\else\\\ifx9########1##5%
3419        \\\else########1%
3420        \\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi
3421        \\\expandafter\<bbl@digits@\languagename>%
3422      \\\fi}}}%
3423   \bbl@tempa}
```

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

```
3424 \def\bbl@buildifcase#1 {% Returns \bbl@tempa, requires \toks@={}
3425   \ifx\\#1%              % \\ before, in case #1 is multiletter
3426     \bbl@exp{%
3427       \def\\\bbl@tempa####1{%
3428         \<ifcase>####1\space\the\toks@\<else>\\\@ctrerr\<fi>}}%
3429   \else
3430     \toks@\expandafter{\the\toks@\or #1}%
3431     \expandafter\bbl@buildifcase
3432   \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).

```
3433 \newcommand\localenumeral[2]{\bbl@cs{cntr@#1@\languagename}{#2}}
3434 \def\bbl@localecntr#1#2{\localenumeral{#2}{#1}}
3435 \newcommand\localecounter[2]{%
3436   \expandafter\bbl@localecntr
3437   \expandafter{\number\csname c@#2\endcsname}{#1}}
3438 \def\bbl@alphnumeral#1#2{%
3439   \expandafter\bbl@alphnumeral@i\number#2 76543210\@@{#1}}
3440 \def\bbl@alphnumeral@i#1#2#3#4#5#6#7#8\@@#9{%
3441   \ifcase\@car#8\@nil\or   % Currenty <10000, but prepared for bigger
3442     \bbl@alphnumeral@ii{#9}000000#1\or
3443     \bbl@alphnumeral@ii{#9}00000#1#2\or
3444     \bbl@alphnumeral@ii{#9}0000#1#2#3\or
3445     \bbl@alphnumeral@ii{#9}000#1#2#3#4\else
3446     \bbl@alphnum@invalid{>9999}%
3447   \fi}
3448 \def\bbl@alphnumeral@ii#1#2#3#4#5#6#7#8{%
3449   \bbl@ifunset{bbl@cntr@#1.F.\number#5#6#7#8@\languagename}%
3450     {\bbl@cs{cntr@#1.4@\languagename}#5%
3451      \bbl@cs{cntr@#1.3@\languagename}#6%
3452      \bbl@cs{cntr@#1.2@\languagename}#7%
3453      \bbl@cs{cntr@#1.1@\languagename}#8%
3454      \ifnum#6#7#8>\z@ % TODO. An ad hoc rule for Greek. Ugly.
3455        \bbl@ifunset{bbl@cntr@#1.S.321@\languagename}{}%
3456          {\bbl@cs{cntr@#1.S.321@\languagename}}%
3457      \fi}%
3458     {\bbl@cs{cntr@#1.F.\number#5#6#7#8@\languagename}}}
3459 \def\bbl@alphnum@invalid#1{%
3460   \bbl@error{Alphabetic numeral too large (#1)}%
3461     {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.

```
3462 \def\bbl@localeinfo#1#2{%
3463   \bbl@ifunset{bbl@info@#2}{#1}%
3464     {\bbl@ifunset{bbl@\csname bbl@info@#2\endcsname @\languagename}{#1}%
3465       {\bbl@cs{\csname bbl@info@#2\endcsname @\languagename}}}}
3466 \newcommand\localeinfo[1]{%
3467   \ifx*#1\@empty   % TODO. A bit hackish to make it expandable.
3468     \bbl@afterelse\bbl@localeinfo{}%
```

```
3469    \else
3470      \bbl@localeinfo
3471        {\bbl@error{I've found no info for the current locale.\\%
3472                  The corresponding ini file has not been loaded\\%
3473                  Perhaps it doesn't exist}%
3474                  {See the manual for details.}}%
3475        {#1}%
3476    \fi}
3477 % \@namedef{bbl@info@name.locale}{lcname}
3478 \@namedef{bbl@info@tag.ini}{lini}
3479 \@namedef{bbl@info@name.english}{elname}
3480 \@namedef{bbl@info@name.opentype}{lname}
3481 \@namedef{bbl@info@tag.bcp47}{tbcp}
3482 \@namedef{bbl@info@language.tag.bcp47}{lbcp}
3483 \@namedef{bbl@info@tag.opentype}{lotf}
3484 \@namedef{bbl@info@script.name}{esname}
3485 \@namedef{bbl@info@script.name.opentype}{sname}
3486 \@namedef{bbl@info@script.tag.bcp47}{sbcp}
3487 \@namedef{bbl@info@script.tag.opentype}{sotf}
3488 \@namedef{bbl@info@region.tag.bcp47}{rbcp}
3489 \@namedef{bbl@info@variant.tag.bcp47}{vbcp}
3490 \@namedef{bbl@info@extension.t.tag.bcp47}{extt}
3491 \@namedef{bbl@info@extension.u.tag.bcp47}{extu}
3492 \@namedef{bbl@info@extension.x.tag.bcp47}{extx}
```

LaTeX needs to know the BCP 47 codes for some features. For that, it expects \BCPdata to be defined. While language, region, script, and variant are recognized, extension.⟨s⟩ for singletons may change.

```
3493 \providecommand\BCPdata{}
3494 \ifx\renewcommand\@undefined\else % For plain. TODO. It's a quick fix
3495    \renewcommand\BCPdata[1]{%
3496      \bbl@ifunset{bbl@info@#1.tag.bcp47}%
3497        {\bbl@error{Unknown field '#1' in \string\BCPdata.\\%
3498                  Perhaps you misspelled it.}%
3499                  {See the manual for details.}}%
3500        {\bbl@ifunset{bbl@\csname bbl@info@#1.tag.bcp47\endcsname @\languagename}{}%
3501          {\bbl@cs{\csname bbl@info@#1.tag.bcp47\endcsname @\languagename}}}}
3502 \fi
```

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

```
3503 ⟨⟨*More package options⟩⟩ ≡
3504 \DeclareOption{ensureinfo=off}{}
3505 ⟨⟨/More package options⟩⟩
3506 %
3507 \let\bbl@ensureinfo\@gobble
3508 \newcommand\BabelEnsureInfo{%
3509    \ifx\InputIfFileExists\@undefined\else
3510      \def\bbl@ensureinfo##1{%
3511        \bbl@ifunset{bbl@lname@##1}{\bbl@load@info{##1}}{}}%
3512    \fi
3513    \bbl@foreach\bbl@loaded{{%
3514      \def\languagename{##1}%
3515      \bbl@ensureinfo{##1}}}}
3516 \@ifpackagewith{babel}{ensureinfo=off}{}%
3517    {\AtEndOfPackage{% Test for plain.
3518      \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.

```
3519 \newcommand\getlocaleproperty{%
3520    \@ifstar\bbl@getproperty@s\bbl@getproperty@x}
3521 \def\bbl@getproperty@s#1#2#3{%
3522    \let#1\relax
```

134

```
3523    \def\bbl@elt##1##2##3{%
3524      \bbl@ifsamestring{##1/##2}{#3}%
3525        {\providecommand#1{##3}%
3526         \def\bbl@elt####1####2####3{}}%
3527        {}}%
3528    \bbl@cs{inidata@#2}}%
3529  \def\bbl@getproperty@x#1#2#3{%
3530    \bbl@getproperty@s{#1}{#2}{#3}%
3531    \ifx#1\relax
3532      \bbl@error
3533        {Unknown key for locale '#2':\\%
3534         #3\\%
3535         \string#1 will be set to \relax}%
3536        {Perhaps you misspelled it.}%
3537    \fi}
3538  \let\bbl@ini@loaded\@empty
3539  \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.

```
3540  \newcommand\babeladjust[1]{%  TODO. Error handling.
3541    \bbl@forkv{#1}{%
3542      \bbl@ifunset{bbl@ADJ@##1@##2}%
3543        {\bbl@cs{ADJ@##1}{##2}}%
3544        {\bbl@cs{ADJ@##1@##2}}}}
3545  %
3546  \def\bbl@adjust@lua#1#2{%
3547    \ifvmode
3548      \ifnum\currentgrouplevel=\z@
3549        \directlua{ Babel.#2 }%
3550        \expandafter\expandafter\expandafter\@gobble
3551      \fi
3552    \fi
3553    {\bbl@error   % The error is gobbled if everything went ok.
3554      {Currently, #1 related features can be adjusted only\\%
3555       in the main vertical list.}%
3556      {Maybe things change in the future, but this is what it is.}}}
3557  \@namedef{bbl@ADJ@bidi.mirroring@on}{%
3558    \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
3559  \@namedef{bbl@ADJ@bidi.mirroring@off}{%
3560    \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
3561  \@namedef{bbl@ADJ@bidi.text@on}{%
3562    \bbl@adjust@lua{bidi}{bidi_enabled=true}}
3563  \@namedef{bbl@ADJ@bidi.text@off}{%
3564    \bbl@adjust@lua{bidi}{bidi_enabled=false}}
3565  \@namedef{bbl@ADJ@bidi.math@on}{%
3566    \let\bbl@noamsmath\@empty}
3567  \@namedef{bbl@ADJ@bidi.math@off}{%
3568    \let\bbl@noamsmath\relax}
3569  \@namedef{bbl@ADJ@bidi.mapdigits@on}{%
3570    \bbl@adjust@lua{bidi}{digits_mapped=true}}
3571  \@namedef{bbl@ADJ@bidi.mapdigits@off}{%
3572    \bbl@adjust@lua{bidi}{digits_mapped=false}}
3573  %
3574  \@namedef{bbl@ADJ@linebreak.sea@on}{%
3575    \bbl@adjust@lua{linebreak}{sea_enabled=true}}
3576  \@namedef{bbl@ADJ@linebreak.sea@off}{%
3577    \bbl@adjust@lua{linebreak}{sea_enabled=false}}
3578  \@namedef{bbl@ADJ@linebreak.cjk@on}{%
3579    \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
3580  \@namedef{bbl@ADJ@linebreak.cjk@off}{%
```

```
3581    \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
3582 \@namedef{bbl@ADJ@justify.arabic@on}{%
3583    \bbl@adjust@lua{linebreak}{arabic.justify_enabled=true}}
3584 \@namedef{bbl@ADJ@justify.arabic@off}{%
3585    \bbl@adjust@lua{linebreak}{arabic.justify_enabled=false}}
3586 %
3587 \def\bbl@adjust@layout#1{%
3588    \ifvmode
3589      #1%
3590      \expandafter\@gobble
3591    \fi
3592    {\bbl@error    % The error is gobbled if everything went ok.
3593       {Currently, layout related features can be adjusted only\\%
3594        in vertical mode.}%
3595       {Maybe things change in the future, but this is what it is.}}}
3596 \@namedef{bbl@ADJ@layout.tabular@on}{%
3597    \ifnum\bbl@tabular@mode=\tw@
3598      \bbl@adjust@layout{\let\@tabular\bbl@NL@@tabular}%
3599    \else
3600      \chardef\bbl@tabular@mode\@ne
3601    \fi}
3602 \@namedef{bbl@ADJ@layout.tabular@off}{%
3603    \ifnum\bbl@tabular@mode=\tw@
3604      \bbl@adjust@layout{\let\@tabular\bbl@OL@@tabular}%
3605    \else
3606      \chardef\bbl@tabular@mode\z@
3607    \fi}
3608 \@namedef{bbl@ADJ@layout.lists@on}{%
3609    \bbl@adjust@layout{\let\list\bbl@NL@list}}
3610 \@namedef{bbl@ADJ@layout.lists@off}{%
3611    \bbl@adjust@layout{\let\list\bbl@OL@list}}
3612 %
3613 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
3614    \bbl@bcpallowedtrue}
3615 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
3616    \bbl@bcpallowedfalse}
3617 \@namedef{bbl@ADJ@autoload.bcp47.prefix}#1{%
3618    \def\bbl@bcp@prefix{#1}}
3619 \def\bbl@bcp@prefix{bcp47-}
3620 \@namedef{bbl@ADJ@autoload.options}#1{%
3621    \def\bbl@autoload@options{#1}}
3622 \let\bbl@autoload@bcpoptions\@empty
3623 \@namedef{bbl@ADJ@autoload.bcp47.options}#1{%
3624    \def\bbl@autoload@bcpoptions{#1}}
3625 \newif\ifbbl@bcptoname
3626 \@namedef{bbl@ADJ@bcp47.toname@on}{%
3627    \bbl@bcptonametrue
3628    \BabelEnsureInfo}
3629 \@namedef{bbl@ADJ@bcp47.toname@off}{%
3630    \bbl@bcptonamefalse}
3631 \@namedef{bbl@ADJ@prehyphenation.disable@nohyphenation}{%
3632    \directlua{ Babel.ignore_pre_char = function(node)
3633       return (node.lang == \the\csname l@nohyphenation\endcsname)
3634      end }}
3635 \@namedef{bbl@ADJ@prehyphenation.disable@off}{%
3636    \directlua{ Babel.ignore_pre_char = function(node)
3637       return false
3638      end }}
3639 \@namedef{bbl@ADJ@select.write@shift}{%
3640    \let\bbl@restorelastskip\relax
3641    \def\bbl@savelastskip{%
3642      \let\bbl@restorelastskip\relax
3643      \ifvmode
```

```
3644      \ifdim\lastskip=\z@
3645        \let\bbl@restorelastskip\nobreak
3646      \else
3647        \bbl@exp{%
3648          \def\\\bbl@restorelastskip{%
3649            \skip@=\the\lastskip
3650            \\\nobreak \vskip-\skip@ \vskip\skip@}}%
3651      \fi
3652    \fi}}
3653 \@namedef{bbl@ADJ@select.write@keep}{%
3654   \let\bbl@restorelastskip\relax
3655   \let\bbl@savelastskip\relax}
3656 \@namedef{bbl@ADJ@select.write@omit}{%
3657   \AddBabelHook{babel-select}{beforestart}{%
3658     \expandafter\babel@aux\expandafter{\bbl@main@language}{}}%
3659   \let\bbl@restorelastskip\relax
3660   \def\bbl@savelastskip##1\bbl@restorelastskip{}}
3661 \@namedef{bbl@ADJ@select.encoding@off}{%
3662   \let\bbl@encoding@select@off\@empty}
```

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

```
3663 \ifx\directlua\@undefined\else
3664   \ifx\bbl@luapatterns\@undefined
3665     \input luababel.def
3666   \fi
3667 \fi
```

Continue with LaTeX.

```
3668 ⟨/package | core⟩
3669 ⟨∗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.

```
3670 ⟨⟨∗More package options⟩⟩ ≡
3671 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
3672 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
3673 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
3674 \DeclareOption{safe=refbib}{\def\bbl@opt@safe{BR}}
3675 \DeclareOption{safe=bibref}{\def\bbl@opt@safe{BR}}
3676 ⟨⟨/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.

```
3677 \bbl@trace{Cross referencing macros}
3678 \ifx\bbl@opt@safe\@empty\else % ie, if 'ref' and/or 'bib'
3679   \def\@newl@bel#1#2#3{%
3680     {\@safe@activestrue
3681      \bbl@ifunset{#1@#2}%
3682        \relax
3683        {\gdef\@multiplelabels{%
3684           \@latex@warning@no@line{There were multiply-defined labels}}%
3685         \@latex@warning@no@line{Label `#2' multiply defined}}%
3686      \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.

```
3687  \CheckCommand*\@testdef[3]{%
3688    \def\reserved@a{#3}%
3689    \expandafter\ifx\csname#1@#2\endcsname\reserved@a
3690    \else
3691      \@tempswatrue
3692    \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.

```
3693  \def\@testdef#1#2#3{%  TODO. With @samestring?
3694    \@safe@activestrue
3695    \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
3696    \def\bbl@tempb{#3}%
3697    \@safe@activesfalse
3698    \ifx\bbl@tempa\relax
3699    \else
3700      \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
3701    \fi
3702    \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
3703    \ifx\bbl@tempa\bbl@tempb
3704    \else
3705      \@tempswatrue
3706    \fi}
3707 \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.

```
3708 \bbl@xin@{R}\bbl@opt@safe
3709 \ifin@
3710  \edef\bbl@tempc{\expandafter\string\csname ref code\endcsname}%
3711  \bbl@xin@{\expandafter\strip@prefix\meaning\bbl@tempc}%
3712    {\expandafter\strip@prefix\meaning\ref}%
3713  \ifin@
3714    \bbl@redefine\@kernel@ref#1{%
3715      \@safe@activestrue\org@@kernel@ref{#1}\@safe@activesfalse}
3716    \bbl@redefine\@kernel@pageref#1{%
3717      \@safe@activestrue\org@@kernel@pageref{#1}\@safe@activesfalse}
3718    \bbl@redefine\@kernel@sref#1{%
3719      \@safe@activestrue\org@@kernel@sref{#1}\@safe@activesfalse}
3720    \bbl@redefine\@kernel@spageref#1{%
3721      \@safe@activestrue\org@@kernel@spageref{#1}\@safe@activesfalse}
3722  \else
3723    \bbl@redefinerobust\ref#1{%
3724      \@safe@activestrue\org@ref{#1}\@safe@activesfalse}
3725    \bbl@redefinerobust\pageref#1{%
3726      \@safe@activestrue\org@pageref{#1}\@safe@activesfalse}
3727  \fi
3728 \else
3729  \let\org@ref\ref
3730  \let\org@pageref\pageref
3731 \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.

```
3732 \bbl@xin@{B}\bbl@opt@safe
3733 \ifin@
3734   \bbl@redefine\@citex[#1]#2{%
3735     \@safe@activestrue\edef\@tempa{#2}\@safe@activesfalse
3736     \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.

```
3737   \AtBeginDocument{%
3738     \@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.)

```
3739     \def\@citex[#1][#2]#3{%
3740       \@safe@activestrue\edef\@tempa{#3}\@safe@activesfalse
3741       \org@@citex[#1][#2]{\@tempa}}%
3742     }{}}
```

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

```
3743   \AtBeginDocument{%
3744     \@ifpackageloaded{cite}{%
3745       \def\@citex[#1]#2{%
3746         \@safe@activestrue\org@@citex[#1]{#2}\@safe@activesfalse}%
3747       }{}}
```

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

```
3748 \bbl@redefine\nocite#1{%
3749   \@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.

```
3750 \bbl@redefine\bibcite{%
3751   \bbl@cite@choice
3752   \bibcite}
```

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

```
3753 \def\bbl@bibcite#1#2{%
3754   \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.

```
3755 \def\bbl@cite@choice{%
3756   \global\let\bibcite\bbl@bibcite
3757   \@ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{}%
3758   \@ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{}%
3759   \global\let\bbl@cite@choice\relax}
```

When a document is run for the first time, no .aux file is available, and \bibcite will not yet be properly defined. In this case, this has to happen before the document starts.

```
3760 \AtBeginDocument{\bbl@cite@choice}
```

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

```
3761  \bbl@redefine\@bibitem#1{%
3762    \@safe@activestrue\org@@bibitem{#1}\@safe@activesfalse}
3763 \else
3764  \let\org@nocite\nocite
3765  \let\org@@citex\@citex
3766  \let\org@bibcite\bibcite
3767  \let\org@@bibitem\@bibitem
3768 \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.

```
3769 \bbl@trace{Marks}
3770 \IfBabelLayout{sectioning}
3771   {\ifx\bbl@opt@headfoot\@nnil
3772     \g@addto@macro\@resetactivechars{%
3773       \set@typeset@protect
3774       \expandafter\select@language@x\expandafter{\bbl@main@language}%
3775       \let\protect\noexpand
3776       \ifcase\bbl@bidimode\else % Only with bidi. See also above
3777         \edef\thepage{%
3778           \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
3779       \fi}%
3780   \fi}
3781 {\ifbbl@single\else
3782   \bbl@ifunset{markright }\bbl@redefine\bbl@redefinerobust
3783   \markright#1{%
3784     \bbl@ifblank{#1}%
3785       {\org@markright{}}%
3786       {\toks@{#1}%
3787        \bbl@exp{%
3788          \\\org@markright{\\\protect\\\foreignlanguage{\languagename}%
3789            {\\\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.)

```
3790    \ifx\@mkboth\markboth
3791      \def\bbl@tempc{\let\@mkboth\markboth}%
3792    \else
3793      \def\bbl@tempc{}%
3794    \fi
3795    \bbl@ifunset{markboth }\bbl@redefine\bbl@redefinerobust
3796    \markboth#1#2{%
3797      \protected@edef\bbl@tempb##1{%
3798        \protect\foreignlanguage
3799        {\languagename}{\protect\bbl@restore@actives##1}}%
3800      \bbl@ifblank{#1}%
3801        {\toks@{}}%
3802        {\toks@\expandafter{\bbl@tempb{#1}}}%
3803      \bbl@ifblank{#2}%
3804        {\@temptokena{}}%
3805        {\@temptokena\expandafter{\bbl@tempb{#2}}}%
```

```
3806         \bbl@exp{\\\org@markboth{\the\toks@}{\the\@temptokena}}}%
3807         \bbl@tempc
3808     \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.

```
3809 \bbl@trace{Preventing clashes with other packages}
3810 \ifx\org@ref\@undefined\else
3811   \bbl@xin@{R}\bbl@opt@safe
3812   \ifin@
3813     \AtBeginDocument{%
3814       \@ifpackageloaded{ifthen}{%
3815         \bbl@redefine@long\ifthenelse#1#2#3{%
3816           \let\bbl@temp@pref\pageref
3817           \let\pageref\org@pageref
3818           \let\bbl@temp@ref\ref
3819           \let\ref\org@ref
3820           \@safe@activestrue
3821           \org@ifthenelse{#1}%
3822             {\let\pageref\bbl@temp@pref
3823              \let\ref\bbl@temp@ref
3824              \@safe@activesfalse
3825              #2}%
3826             {\let\pageref\bbl@temp@pref
3827              \let\ref\bbl@temp@ref
3828              \@safe@activesfalse
3829              #3}%
3830         }%
3831       }{}%
3832     }
3833 \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.

```
3834   \AtBeginDocument{%
3835     \@ifpackageloaded{varioref}{%
3836       \bbl@redefine\@@vpageref#1[#2]#3{%
3837         \@safe@activestrue
3838         \org@@@vpageref{#1}[#2]{#3}%
3839         \@safe@activesfalse}%
3840       \bbl@redefine\vrefpagenum#1#2{%
3841         \@safe@activestrue
```

```
3842            \org@vrefpagenum{#1}{#2}%
3843            \@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.

```
3844        \expandafter\def\csname Ref \endcsname#1{%
3845          \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
3846        }{}%
3847      }
3848 \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.

```
3849 \AtEndOfPackage{%
3850   \AtBeginDocument{%
3851     \@ifpackageloaded{hhline}%
3852       {\expandafter\ifx\csname normal@char\string:\endcsname\relax
3853        \else
3854          \makeatletter
3855          \def\@currname{hhline}\input{hhline.sty}\makeatother
3856        \fi}%
3857       {}}}
```

\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.

```
3858 \def\substitutefontfamily#1#2#3{%
3859   \lowercase{\immediate\openout15=#1#2.fd\relax}%
3860   \immediate\write15{%
3861     \string\ProvidesFile{#1#2.fd}%
3862     [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
3863      \space generated font description file]^^J
3864     \string\DeclareFontFamily{#1}{#2}{}^^J
3865     \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{}^^J
3866     \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{}^^J
3867     \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{}^^J
3868     \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{}^^J
3869     \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{}^^J
3870     \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{}^^J
3871     \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{}^^J
3872     \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{}^^J
3873     }%
3874   \closeout15
3875   }
3876 \@onlypreamble\substitutefontfamily
```

8.4 Encoding and fonts

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

\ensureascii

```
3877 \bbl@trace{Encoding and fonts}
3878 \newcommand\BabelNonASCII{LGR,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU}
3879 \newcommand\BabelNonText{TS1,T3,TS3}
3880 \let\org@TeX\TeX
3881 \let\org@LaTeX\LaTeX
3882 \let\ensureascii\@firstofone
3883 \AtBeginDocument{%
3884   \def\@elt#1{,#1,}%
3885   \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3886   \let\@elt\relax
3887   \let\bbl@tempb\@empty
3888   \def\bbl@tempc{OT1}%
3889   \bbl@foreach\BabelNonASCII{% LGR loaded in a non-standard way
3890     \bbl@ifunset{T@#1}{}{\def\bbl@tempb{#1}}}%
3891   \bbl@foreach\bbl@tempa{%
3892     \bbl@xin@{#1}{\BabelNonASCII}%
3893     \ifin@
3894       \def\bbl@tempb{#1}% Store last non-ascii
3895     \else\bbl@xin@{#1}{\BabelNonText}% Pass
3896       \ifin@\else
3897         \def\bbl@tempc{#1}% Store last ascii
3898       \fi
3899     \fi}%
3900   \ifx\bbl@tempb\@empty\else
3901     \bbl@xin@{,\cf@encoding,}{,\BabelNonASCII,\BabelNonText,}%
3902     \ifin@\else
3903       \edef\bbl@tempc{\cf@encoding}% The default if ascii wins
3904     \fi
3905     \edef\ensureascii#1{%
3906       {\noexpand\fontencoding{\bbl@tempc}\noexpand\selectfont#1}}%
3907     \DeclareTextCommandDefault{\TeX}{\ensureascii{\org@TeX}}%
3908     \DeclareTextCommandDefault{\LaTeX}{\ensureascii{\org@LaTeX}}%
3909   \fi}
```

Now comes the old deprecated stuff (with a little change in 3.9l, for fontspec). The first thing we need to do is to determine, at \begin{document}, which latin fontencoding to use.

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

```
3910 \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.

```
3911 \AtBeginDocument{%
3912   \@ifpackageloaded{fontspec}%
3913     {\xdef\latinencoding{%
3914        \ifx\UTFencname\@undefined
3915          EU\ifcase\bbl@engine\or2\or1\fi
3916        \else
3917          \UTFencname
3918        \fi}}%
3919     {\gdef\latinencoding{OT1}%
3920      \ifx\cf@encoding\bbl@t@one
3921        \xdef\latinencoding{\bbl@t@one}%
3922      \else
3923        \def\@elt#1{,#1,}%
3924        \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3925        \let\@elt\relax
3926        \bbl@xin@{,T1,}\bbl@tempa
```

143

```
3927        \ifin@
3928          \xdef\latinencoding{\bbl@t@one}%
3929        \fi
3930      \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.

```
3931 \DeclareRobustCommand{\latintext}{%
3932   \fontencoding{\latinencoding}\selectfont
3933   \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.

```
3934 \ifx\@undefined\DeclareTextFontCommand
3935   \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
3936 \else
3937   \DeclareTextFontCommand{\textlatin}{\latintext}
3938 \fi
```

For several functions, we need to execute some code with \selectfont. With LATEX 2021-06-01, there is a hook for this purpose.

```
3939 \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.

```
3940 \bbl@trace{Loading basic (internal) bidi support}
3941 \ifodd\bbl@engine
3942 \else % TODO. Move to txtbabel
3943   \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
3944     \bbl@error
3945       {The bidi method 'basic' is available only in\\%
3946        luatex. I'll continue with 'bidi=default', so\\%
3947        expect wrong results}%
3948       {See the manual for further details.}%
3949     \let\bbl@beforeforeign\leavevmode
3950     \AtEndOfPackage{%
3951       \EnableBabelHook{babel-bidi}%
3952       \bbl@xebidipar}
3953   \fi\fi
3954   \def\bbl@loadxebidi#1{%
3955     \ifx\RTLfootnotetext\@undefined
3956       \AtEndOfPackage{%
```

```
3957        \EnableBabelHook{babel-bidi}%
3958        \bbl@loadfontspec % bidi needs fontspec
3959        \usepackage#1{bidi}}%
3960     \fi}
3961   \ifnum\bbl@bidimode>200
3962     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
3963       \bbl@tentative{bidi=bidi}
3964       \bbl@loadxebidi{}
3965     \or
3966       \bbl@loadxebidi{[rldocument]}
3967     \or
3968       \bbl@loadxebidi{}
3969     \fi
3970   \fi
3971 \fi
3972 % TODO? Separate:
3973 \ifnum\bbl@bidimode=\@ne
3974   \let\bbl@beforeforeign\leavevmode
3975   \ifodd\bbl@engine
3976     \newattribute\bbl@attr@dir
3977     \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
3978     \bbl@exp{\output{\bodydir\pagedir\the\output}}
3979   \fi
3980   \AtEndOfPackage{%
3981     \EnableBabelHook{babel-bidi}%
3982     \ifodd\bbl@engine\else
3983       \bbl@xebidipar
3984     \fi}
3985 \fi
```

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

```
3986 \bbl@trace{Macros to switch the text direction}
3987 \def\bbl@alscripts{,Arabic,Syriac,Thaana,}
3988 \def\bbl@rscripts{% TODO. Base on codes ??
3989   ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
3990   Old Hungarian,Lydian,Mandaean,Manichaean,%
3991   Meroitic Cursive,Meroitic,Old North Arabian,%
3992   Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
3993   Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
3994   Old South Arabian,}%
3995 \def\bbl@provide@dirs#1{%
3996   \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
3997   \ifin@
3998     \global\bbl@csarg\chardef{wdir@#1}\@ne
3999     \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
4000     \ifin@
4001       \global\bbl@csarg\chardef{wdir@#1}\tw@  % useless in xetex
4002     \fi
4003   \else
4004     \global\bbl@csarg\chardef{wdir@#1}\z@
4005   \fi
4006   \ifodd\bbl@engine
4007     \bbl@csarg\ifcase{wdir@#1}%
4008       \directlua{ Babel.locale_props[\the\localeid].textdir = 'l' }%
4009     \or
4010       \directlua{ Babel.locale_props[\the\localeid].textdir = 'r' }%
4011     \or
4012       \directlua{ Babel.locale_props[\the\localeid].textdir = 'al' }%
4013     \fi
4014   \fi}
4015 \def\bbl@switchdir{%
4016   \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
```

```
4017    \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
4018    \bbl@exp{\\\bbl@setdirs\bbl@cl{wdir}}}}
4019 \def\bbl@setdirs#1{% TODO - math
4020    \ifcase\bbl@select@type % TODO - strictly, not the right test
4021      \bbl@bodydir{#1}%
4022      \bbl@pardir{#1}% <- Must precede \bbl@textdir
4023    \fi
4024    \bbl@textdir{#1}}
4025 % TODO. Only if \bbl@bidimode > 0?:
4026 \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
4027 \DisableBabelHook{babel-bidi}
```

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

```
4028 \ifodd\bbl@engine  % luatex=1
4029 \else % pdftex=0, xetex=2
4030    \newcount\bbl@dirlevel
4031    \chardef\bbl@thetextdir\z@
4032    \chardef\bbl@thepardir\z@
4033    \def\bbl@textdir#1{%
4034      \ifcase#1\relax
4035        \chardef\bbl@thetextdir\z@
4036        \bbl@textdir@i\beginL\endL
4037      \else
4038        \chardef\bbl@thetextdir\@ne
4039        \bbl@textdir@i\beginR\endR
4040      \fi}
4041    \def\bbl@textdir@i#1#2{%
4042      \ifhmode
4043        \ifnum\currentgrouplevel>\z@
4044          \ifnum\currentgrouplevel=\bbl@dirlevel
4045            \bbl@error{Multiple bidi settings inside a group}%
4046              {I'll insert a new group, but expect wrong results.}%
4047            \bgroup\aftergroup#2\aftergroup\egroup
4048          \else
4049            \ifcase\currentgrouptype\or % 0 bottom
4050              \aftergroup#2% 1 simple {}
4051            \or
4052              \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
4053            \or
4054              \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
4055            \or\or\or % vbox vtop align
4056            \or
4057              \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
4058            \or\or\or\or\or\or % output math disc insert vcent mathchoice
4059            \or
4060              \aftergroup#2% 14 \begingroup
4061            \else
4062              \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
4063            \fi
4064          \fi
4065          \bbl@dirlevel\currentgrouplevel
4066        \fi
4067        #1%
4068      \fi}
4069    \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
4070    \let\bbl@bodydir\@gobble
4071    \let\bbl@pagedir\@gobble
4072    \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).

```
4073    \def\bbl@xebidipar{%
4074      \let\bbl@xebidipar\relax
```

```
4075    \TeXXeTstate\@ne
4076    \def\bbl@xeeverypar{%
4077      \ifcase\bbl@thepardir
4078        \ifcase\bbl@thetextdir\else\beginR\fi
4079      \else
4080        {\setbox\z@\lastbox\beginR\box\z@}%
4081      \fi}%
4082    \let\bbl@severypar\everypar
4083    \newtoks\everypar
4084    \everypar=\bbl@severypar
4085    \bbl@severypar{\bbl@xeeverypar\the\everypar}}
4086  \ifnum\bbl@bidimode>200
4087    \let\bbl@textdir@i\@gobbletwo
4088    \let\bbl@xebidipar\@empty
4089    \AddBabelHook{bidi}{foreign}{%
4090      \def\bbl@tempa{\def\BabelText####1}%
4091      \ifcase\bbl@thetextdir
4092        \expandafter\bbl@tempa\expandafter{\BabelText{\LR{##1}}}%
4093      \else
4094        \expandafter\bbl@tempa\expandafter{\BabelText{\RL{##1}}}%
4095      \fi}
4096    \def\bbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}
4097  \fi
4098 \fi
```

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

```
4099 \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}
4100 \AtBeginDocument{%
4101  \ifx\pdfstringdefDisableCommands\@undefined\else
4102    \ifx\pdfstringdefDisableCommands\relax\else
4103      \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
4104    \fi
4105  \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.

```
4106 \bbl@trace{Local Language Configuration}
4107 \ifx\loadlocalcfg\@undefined
4108  \@ifpackagewith{babel}{noconfigs}%
4109    {\let\loadlocalcfg\@gobble}%
4110    {\def\loadlocalcfg#1{%
4111      \InputIfFileExists{#1.cfg}%
4112        {\typeout{*************************************^^J%
4113                      * Local config file #1.cfg used^^J%
4114                      *}}%
4115      \@empty}}
4116 \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).

```
4117 \bbl@trace{Language options}
4118 \let\bbl@afterlang\relax
4119 \let\BabelModifiers\relax
4120 \let\bbl@loaded\@empty
```

```
4121 \def\bbl@load@language#1{%
4122   \InputIfFileExists{#1.ldf}%
4123     {\edef\bbl@loaded{\CurrentOption
4124        \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}%
4125      \expandafter\let\expandafter\bbl@afterlang
4126        \csname\CurrentOption.ldf-h@@k\endcsname
4127      \expandafter\let\expandafter\BabelModifiers
4128        \csname bbl@mod@\CurrentOption\endcsname
4129      \bbl@exp{\\\AtBeginDocument{%
4130        \\\bbl@usehooks@lang{\CurrentOption}{begindocument}{{\CurrentOption}}}}}%
4131     {\bbl@error{%
4132        Unknown option '\CurrentOption'. Either you misspelled it\\%
4133        or the language definition file \CurrentOption.ldf was not found}{%
4134        Valid options are, among others: shorthands=, KeepShorthandsActive,\\%
4135        activeacute, activegrave, noconfigs, safe=, main=, math=\\%
4136        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.

```
4137 \def\bbl@try@load@lang#1#2#3{%
4138   \IfFileExists{\CurrentOption.ldf}%
4139     {\bbl@load@language{\CurrentOption}}%
4140     {#1\bbl@load@language{#2}#3}}
4141 %
4142 \DeclareOption{hebrew}{%
4143   \input{rlbabel.def}%
4144   \bbl@load@language{hebrew}}
4145 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}{}}
4146 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}{}}
4147 \DeclareOption{nynorsk}{\bbl@try@load@lang{}{norsk}{}}
4148 \DeclareOption{polutonikogreek}{%
4149   \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}
4150 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}{}}
4151 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}{}}
4152 \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.

```
4153 \ifx\bbl@opt@config\@nnil
4154   \@ifpackagewith{babel}{noconfigs}{}%
4155     {\InputIfFileExists{bblopts.cfg}%
4156        {\typeout{*************************************^^J%
4157                 * Local config file bblopts.cfg used^^J%
4158                 *}}%
4159        {}}%
4160 \else
4161   \InputIfFileExists{\bbl@opt@config.cfg}%
4162     {\typeout{*************************************^^J%
4163                 * Local config file \bbl@opt@config.cfg used^^J%
4164                 *}}%
4165     {\bbl@error{%
4166        Local config file '\bbl@opt@config.cfg' not found}{%
4167        Perhaps you misspelled it.}}%
4168 \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.

```
4169 \ifx\bbl@opt@main\@nnil
4170   \ifnum\bbl@iniflag>\z@  % if all ldf's: set implicitly, no main pass
4171     \let\bbl@tempb\@empty
4172     \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}%
4173     \bbl@foreach\bbl@tempa{\edef\bbl@tempb{#1,\bbl@tempb}}%
4174     \bbl@foreach\bbl@tempb{%    \bbl@tempb is a reversed list
4175       \ifx\bbl@opt@main\@nnil % ie, if not yet assigned
4176         \ifodd\bbl@iniflag % = *=
4177           \IfFileExists{babel-#1.tex}{\def\bbl@opt@main{#1}}{}%
4178         \else % n +=
4179           \IfFileExists{#1.ldf}{\def\bbl@opt@main{#1}}{}%
4180         \fi
4181       \fi}%
4182   \fi
4183 \else
4184   \bbl@info{Main language set with 'main='. Except if you have\\%
4185            problems, prefer the default mechanism for setting\\%
4186            the main language, ie, as the last declared.\\%
4187            Reported}
4188 \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).

```
4189 \ifx\bbl@opt@main\@nnil\else
4190   \bbl@ncarg\let\bbl@loadmain{ds@\bbl@opt@main}%
4191   \expandafter\let\csname ds@\bbl@opt@main\endcsname\relax
4192 \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.

```
4193 \bbl@foreach\bbl@language@opts{%
4194   \def\bbl@tempa{#1}%
4195   \ifx\bbl@tempa\bbl@opt@main\else
4196     \ifnum\bbl@iniflag<\tw@    % 0 ø (other = ldf)
4197       \bbl@ifunset{ds@#1}%
4198         {\DeclareOption{#1}{\bbl@load@language{#1}}}%
4199         {}%
4200     \else                      % + * (other = ini)
4201       \DeclareOption{#1}{%
4202         \bbl@ldfinit
4203         \babelprovide[import]{#1}%
4204         \bbl@afterldf{}}%
4205     \fi
4206   \fi}
4207 \bbl@foreach\@classoptionslist{%
4208   \def\bbl@tempa{#1}%
4209   \ifx\bbl@tempa\bbl@opt@main\else
4210     \ifnum\bbl@iniflag<\tw@    % 0 ø (other = ldf)
4211       \bbl@ifunset{ds@#1}%
4212         {\IfFileExists{#1.ldf}%
4213           {\DeclareOption{#1}{\bbl@load@language{#1}}}%
4214           {}}%
4215         {}%
4216     \else                      % + * (other = ini)
4217       \IfFileExists{babel-#1.tex}%
4218         {\DeclareOption{#1}{%
4219           \bbl@ldfinit
4220           \babelprovide[import]{#1}%
4221           \bbl@afterldf{}}}%
4222         {}%
4223     \fi
```

```
4224    \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):

```
4225 \def\AfterBabelLanguage#1{%
4226   \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang}{}}
4227 \DeclareOption*{}
4228 \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.

```
4229 \bbl@trace{Option 'main'}
4230 \ifx\bbl@opt@main\@nnil
4231   \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}
4232   \let\bbl@tempc\@empty
4233   \edef\bbl@templ{,\bbl@loaded,}
4234   \edef\bbl@templ{\expandafter\strip@prefix\meaning\bbl@templ}
4235   \bbl@for\bbl@tempb\bbl@tempa{%
4236     \edef\bbl@tempd{,\bbl@tempb,}%
4237     \edef\bbl@tempd{\expandafter\strip@prefix\meaning\bbl@tempd}%
4238     \bbl@xin@{\bbl@tempd}{\bbl@templ}%
4239     \ifin@\edef\bbl@tempc{\bbl@tempb}\fi}
4240   \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
4241   \expandafter\bbl@tempa\bbl@loaded,\@nnil
4242   \ifx\bbl@tempb\bbl@tempc\else
4243     \bbl@warning{%
4244       Last declared language option is '\bbl@tempc',\\%
4245       but the last processed one was '\bbl@tempb'.\\%
4246       The main language can't be set as both a global\\%
4247       and a package option. Use 'main=\bbl@tempc' as\\%
4248       option. Reported}
4249   \fi
4250 \else
4251   \ifodd\bbl@iniflag  % case 1,3 (main is ini)
4252     \bbl@ldfinit
4253     \let\CurrentOption\bbl@opt@main
4254     \bbl@exp{%  \bbl@opt@provide = empty if *
4255       \\\babelprovide[\bbl@opt@provide,import,main]{\bbl@opt@main}}%
4256     \bbl@afterldf{}
4257     \DeclareOption{\bbl@opt@main}{}
4258   \else % case 0,2 (main is ldf)
4259     \ifx\bbl@loadmain\relax
4260       \DeclareOption{\bbl@opt@main}{\bbl@load@language{\bbl@opt@main}}
4261     \else
4262       \DeclareOption{\bbl@opt@main}{\bbl@loadmain}
4263     \fi
4264     \ExecuteOptions{\bbl@opt@main}
4265     \@namedef{ds@\bbl@opt@main}{}%
4266   \fi
4267   \DeclareOption*{}
4268   \ProcessOptions*
4269 \fi
4270 \bbl@exp{% TODO. Find a 'language name' for the end
4271   \\\AtBeginDocument{\\\bbl@usehooks@lang{/}{begindocument}{{}}}}%
4272 \def\AfterBabelLanguage{%
4273   \bbl@error
4274     {Too late for \string\AfterBabelLanguage}%
4275     {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.

```
4276 \ifx\bbl@main@language\@undefined
4277   \bbl@info{%
4278     You haven't specified a language as a class or package\\%
4279     option. I'll load 'nil'. Reported}
4280     \bbl@load@language{nil}
4281 \fi
4282 ⟨/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

```
4283 ⟨*kernel⟩
4284 \let\bbl@onlyswitch\@empty
4285 \input babel.def
4286 \let\bbl@onlyswitch\@undefined
4287 ⟨/kernel⟩
4288 ⟨*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.

```
4289 ⟨⟨Make sure ProvidesFile is defined⟩⟩
4290 \ProvidesFile{hyphen.cfg}[⟨⟨date⟩⟩ ⟨⟨version⟩⟩ Babel hyphens]
4291 \xdef\bbl@format{\jobname}
4292 \def\bbl@version{⟨⟨version⟩⟩}
4293 \def\bbl@date{⟨⟨date⟩⟩}
4294 \ifx\AtBeginDocument\@undefined
4295   \def\@empty{}
4296 \fi
4297 ⟨⟨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.

```
4298 \def\process@line#1#2 #3 #4 {%
4299   \ifx=#1%
4300     \process@synonym{#2}%
4301   \else
4302     \process@language{#1#2}{#3}{#4}%
4303   \fi
4304   \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.

```
4305 \toks@{}
4306 \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.

```
4307 \def\process@synonym#1{%
4308   \ifnum\last@language=\m@ne
4309     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4310   \else
4311     \expandafter\chardef\csname l@#1\endcsname\last@language
4312     \wlog{\string\l@#1=\string\language\the\last@language}%
4313     \expandafter\let\csname #1hyphenmins\expandafter\endcsname
4314       \csname\languagename hyphenmins\endcsname
4315     \let\bbl@elt\relax
4316     \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}{}{}}%
4317   \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.

```
4318 \def\process@language#1#2#3{%
4319   \expandafter\addlanguage\csname l@#1\endcsname
4320   \expandafter\language\csname l@#1\endcsname
4321   \edef\languagename{#1}%
4322   \bbl@hook@everylanguage{#1}%
4323   %  > luatex
4324   \bbl@get@enc#1::\@@@
4325   \begingroup
4326     \lefthyphenmin\m@ne
4327     \bbl@hook@loadpatterns{#2}%
4328     %  > luatex
4329     \ifnum\lefthyphenmin=\m@ne
4330     \else
4331       \expandafter\xdef\csname #1hyphenmins\endcsname{%
4332         \the\lefthyphenmin\the\righthyphenmin}%
4333     \fi
4334   \endgroup
4335   \def\bbl@tempa{#3}%
4336   \ifx\bbl@tempa\@empty\else
```

```
4337    \bbl@hook@loadexceptions{#3}%
4338    % > luatex
4339    \fi
4340   \let\bbl@elt\relax
4341   \edef\bbl@languages{%
4342      \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
4343   \ifnum\the\language=\z@
4344      \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4345         \set@hyphenmins\tw@\thr@@\relax
4346      \else
4347         \expandafter\expandafter\expandafter\set@hyphenmins
4348            \csname #1hyphenmins\endcsname
4349      \fi
4350      \the\toks@
4351      \toks@{}%
4352   \fi}
```

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

```
4353 \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.

```
4354 \def\bbl@hook@everylanguage#1{}
4355 \def\bbl@hook@loadpatterns#1{\input #1\relax}
4356 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
4357 \def\bbl@hook@loadkernel#1{%
4358   \def\addlanguage{\csname newlanguage\endcsname}%
4359   \def\adddialect##1##2{%
4360      \global\chardef##1##2\relax
4361      \wlog{\string##1 = a dialect from \string\language##2}}%
4362   \def\iflanguage##1{%
4363      \expandafter\ifx\csname l@##1\endcsname\relax
4364         \@nolanerr{##1}%
4365      \else
4366         \ifnum\csname l@##1\endcsname=\language
4367            \expandafter\expandafter\expandafter\@firstoftwo
4368         \else
4369            \expandafter\expandafter\expandafter\@secondoftwo
4370         \fi
4371      \fi}%
4372   \def\providehyphenmins##1##2{%
4373      \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
4374         \@namedef{##1hyphenmins}{##2}%
4375      \fi}%
4376   \def\set@hyphenmins##1##2{%
4377      \lefthyphenmin##1\relax
4378      \righthyphenmin##2\relax}%
4379   \def\selectlanguage{%
4380      \errhelp{Selecting a language requires a package supporting it}%
4381      \errmessage{Not loaded}}%
4382   \let\foreignlanguage\selectlanguage
4383   \let\otherlanguage\selectlanguage
4384   \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4385   \def\bbl@usehooks##1##2{}% TODO. Temporary!!
4386   \def\setlocale{%
4387      \errhelp{Find an armchair, sit down and wait}%
4388      \errmessage{Not yet available}}%
4389   \let\uselocale\setlocale
4390   \let\locale\setlocale
4391   \let\selectlocale\setlocale
4392   \let\localename\setlocale
```

```
4393    \let\textlocale\setlocale
4394    \let\textlanguage\setlocale
4395    \let\languagetext\setlocale}
4396  \begingroup
4397  \def\AddBabelHook#1#2{%
4398     \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
4399       \def\next{\toks1}%
4400     \else
4401       \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname####1}%
4402     \fi
4403     \next}
4404  \ifx\directlua\@undefined
4405     \ifx\XeTeXinputencoding\@undefined\else
4406       \input xebabel.def
4407     \fi
4408  \else
4409     \input luababel.def
4410  \fi
4411  \openin1 = babel-\bbl@format.cfg
4412  \ifeof1
4413  \else
4414     \input babel-\bbl@format.cfg\relax
4415  \fi
4416  \closein1
4417  \endgroup
4418  \bbl@hook@loadkernel{switch.def}
```

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

```
4419  \openin1 = language.dat
```

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

```
4420  \def\languagename{english}%
4421  \ifeof1
4422    \message{I couldn't find the file language.dat,\space
4423            I will try the file hyphen.tex}
4424    \input hyphen.tex\relax
4425    \chardef\l@english\z@
4426  \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.

```
4427    \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.

```
4428    \loop
4429      \endlinechar\m@ne
4430      \read1 to \bbl@line
4431      \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.

```
4432      \if T\ifeof1F\fi T\relax
4433        \ifx\bbl@line\@empty\else
4434          \edef\bbl@line{\bbl@line\space\space\space}%
4435          \expandafter\process@line\bbl@line\relax
4436        \fi
4437    \repeat
```

154

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.

```
4438   \begingroup
4439     \def\bbl@elt#1#2#3#4{%
4440       \global\language=#2\relax
4441       \gdef\languagename{#1}%
4442       \def\bbl@elt##1##2##3##4{}}%
4443     \bbl@languages
4444   \endgroup
4445 \fi
4446 \closein1
```

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

```
4447 \if/\the\toks@/\else
4448   \errhelp{language.dat loads no language, only synonyms}
4449   \errmessage{Orphan language synonym}
4450 \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.

```
4451 \let\bbl@line\@undefined
4452 \let\process@line\@undefined
4453 \let\process@synonym\@undefined
4454 \let\process@language\@undefined
4455 \let\bbl@get@enc\@undefined
4456 \let\bbl@hyph@enc\@undefined
4457 \let\bbl@tempa\@undefined
4458 \let\bbl@hook@loadkernel\@undefined
4459 \let\bbl@hook@everylanguage\@undefined
4460 \let\bbl@hook@loadpatterns\@undefined
4461 \let\bbl@hook@loadexceptions\@undefined
4462 ⟨/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].

```
4463 ⟨⟨*More package options⟩⟩ ≡
4464 \chardef\bbl@bidimode\z@
4465 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4466 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
4467 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4468 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4469 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4470 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4471 ⟨⟨/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 (and mostly unuseful) message.

```
4472 ⟨⟨*Font selection⟩⟩ ≡
4473 \bbl@trace{Font handling with fontspec}
4474 \ifx\ExplSyntaxOn\@undefined\else
4475   \def\bbl@fs@warn@nx#1#2{% \bbl@tempfs is the original macro
4476     \in@{,#1,}{,no-script,language-not-exist,}%
4477     \ifin@\else\bbl@tempfs@nx{#1}{#2}\fi}
```

```
4478   \def\bbl@fs@warn@nxx#1#2#3{%
4479     \in@{,#1,}{,no-script,language-not-exist,}%
4480     \ifin@\else\bbl@tempfs@nxx{#1}{#2}{#3}\fi}
4481   \def\bbl@loadfontspec{%
4482     \let\bbl@loadfontspec\relax
4483     \ifx\fontspec\@undefined
4484       \usepackage{fontspec}%
4485     \fi}%
4486 \fi
4487 \@onlypreamble\babelfont
4488 \newcommand\babelfont[2][]{%  1=langs/scripts 2=fam
4489   \bbl@foreach{#1}{%
4490     \expandafter\ifx\csname date##1\endcsname\relax
4491       \IfFileExists{babel-##1.tex}%
4492         {\babelprovide{##1}}%
4493         {}%
4494     \fi}%
4495   \edef\bbl@tempa{#1}%
4496   \def\bbl@tempb{#2}%  Used by \bbl@bblfont
4497   \bbl@loadfontspec
4498   \EnableBabelHook{babel-fontspec}% Just calls \bbl@switchfont
4499   \bbl@bblfont}
4500 \newcommand\bbl@bblfont[2][]{% 1=features 2=fontname, @font=rm|sf|tt
4501   \bbl@ifunset{\bbl@tempb family}%
4502     {\bbl@providefam{\bbl@tempb}}%
4503     {}%
4504   % For the default font, just in case:
4505   \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
4506   \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
4507     {\bbl@csarg\edef{\bbl@tempb dflt@}{<>{#1}{#2}}% save bbl@rmdflt@
4508      \bbl@exp{%
4509        \let\<bbl@\bbl@tempb dflt@\languagename>\<bbl@\bbl@tempb dflt@>%
4510        \\\bbl@font@set\<bbl@\bbl@tempb dflt@\languagename>%
4511                       \<\bbl@tempb default>\<\bbl@tempb family>}}%
4512     {\bbl@foreach\bbl@tempa{% ie bbl@rmdflt@lang / *scrt
4513        \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:

```
4514 \def\bbl@providefam#1{%
4515   \bbl@exp{%
4516     \\\newcommand\<#1default>{}% Just define it
4517     \\\bbl@add@list\\\bbl@font@fams{#1}%
4518     \\\DeclareRobustCommand\<#1family>{%
4519       \\\not@math@alphabet\<#1family>\relax
4520       % \\\prepare@family@series@update{#1}\<#1default>% TODO. Fails
4521       \\\fontfamily\<#1default>%
4522       \<\ifx>\\\UseHooks\\\@undefined\<\else>\\\UseHook{#1family}\<\fi>%
4523       \\\selectfont}%
4524     \\\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.

```
4525 \def\bbl@nostdfont#1{%
4526   \bbl@ifunset{bbl@WFF@\f@family}%
4527     {\bbl@csarg\gdef{WFF@\f@family}{}%  Flag, to avoid dupl warns
4528      \bbl@infowarn{The current font is not a babel standard family:\\%
4529        #1%
4530        \fontname\font\\%
4531        There is nothing intrinsically wrong with this warning, and\\%
4532        you can ignore it altogether if you do not need these\\%
4533        families. But if they are used in the document, you should be\\%
4534        aware 'babel' will not set Script and Language for them, so\\%
4535        you may consider defining a new family with \string\babelfont.\\%
4536        See the manual for further details about \string\babelfont.\\%
```

156

```
4537       Reported}}
4538   {}}%
4539 \gdef\bbl@switchfont{%
4540   \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
4541   \bbl@exp{%  eg Arabic -> arabic
4542     \lowercase{\edef\\\bbl@tempa{\bbl@cl{sname}}}}%
4543   \bbl@foreach\bbl@font@fams{%
4544     \bbl@ifunset{bbl@##1dflt@\languagename}%      (1) language?
4545       {\bbl@ifunset{bbl@##1dflt@*\bbl@tempa}%    (2) from script?
4546         {\bbl@ifunset{bbl@##1dflt@}%              2=F - (3) from generic?
4547           {}%                                   123=F - nothing!
4548           {\bbl@exp{%                            3=T - from generic
4549             \global\let\<bbl@##1dflt@\languagename>%
4550                        \<bbl@##1dflt@>}}}%
4551         {\bbl@exp{%                              2=T - from script
4552           \global\let\<bbl@##1dflt@\languagename>%
4553                      \<bbl@##1dflt@*\bbl@tempa>}}}%
4554       {}}%                                      1=T - language, already defined
4555   \def\bbl@tempa{\bbl@nostdfont{}}%  TODO. Don't use \bbl@tempa
4556   \bbl@foreach\bbl@font@fams{%      don't gather with prev for
4557     \bbl@ifunset{bbl@##1dflt@\languagename}%
4558       {\bbl@cs{famrst@##1}%
4559        \global\bbl@csarg\let{famrst@##1}\relax}%
4560       {\bbl@exp{% order is relevant. TODO: but sometimes wrong!
4561          \\\bbl@add\\\originalTeX{%
4562            \\\bbl@font@rst{\bbl@cl{##1dflt}}%
4563                          \<##1default>\<##1family>{##1}}%
4564          \\\bbl@font@set\<bbl@##1dflt@\languagename>% the main part!
4565                        \<##1default>\<##1family>}}}%
4566   \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.

```
4567 \ifx\f@family\@undefined\else    % if latex
4568   \ifcase\bbl@engine            % if pdftex
4569     \let\bbl@ckeckstdfonts\relax
4570   \else
4571     \def\bbl@ckeckstdfonts{%
4572       \begingroup
4573         \global\let\bbl@ckeckstdfonts\relax
4574         \let\bbl@tempa\@empty
4575         \bbl@foreach\bbl@font@fams{%
4576           \bbl@ifunset{bbl@##1dflt@}%
4577             {\@nameuse{##1family}%
4578              \bbl@csarg\gdef{WFF@\f@family}{}% Flag
4579              \bbl@exp{\\\bbl@add\\\bbl@tempa{* \<##1family>= \f@family\\\\%
4580                \space\space\fontname\font\\\\}}%
4581              \bbl@csarg\xdef{##1dflt@}{\f@family}%
4582              \expandafter\xdef\csname ##1default\endcsname{\f@family}}%
4583             {}}%
4584         \ifx\bbl@tempa\@empty\else
4585           \bbl@infowarn{The following font families will use the default\\%
4586             settings for all or some languages:\\%
4587             \bbl@tempa
4588             There is nothing intrinsically wrong with it, but\\%
4589             'babel' will no set Script and Language, which could\\%
4590             be relevant in some languages. If your document uses\\%
4591             these families, consider redefining them with \string\babelfont.\\%
4592             Reported}%
4593         \fi
4594       \endgroup}
4595   \fi
4596 \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.

```
4597 \def\bbl@font@set#1#2#3{% eg \bbl@rmdflt@lang \rmdefault \rmfamily
4598   \bbl@xin@{<>}{#1}%
4599   \ifin@
4600     \bbl@exp{\\\bbl@fontspec@set\\#1\expandafter\@gobbletwo#1\\#3}%
4601   \fi
4602 \bbl@exp{%                'Unprotected' macros return prev values
4603     \def\\#2{#1}%          eg, \rmdefault{\bbl@rmdflt@lang}
4604     \\\bbl@ifsamestring{#2}{\f@family}%
4605       {\\#3%
4606        \\\bbl@ifsamestring{\f@series}{\bfdefault}{\\\bfseries}{}%
4607        \let\\\bbl@tempa\relax}%
4608       {}}}
4609 %     TODO - next should be global?, but even local does its job. I'm
4610 %     still not sure -- must investigate:
4611 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
4612   \let\bbl@tempe\bbl@mapselect
4613   \let\bbl@mapselect\relax
4614   \let\bbl@temp@fam#4%         eg, '\rmfamily', to be restored below
4615   \let#4\@empty       %        Make sure \renewfontfamily is valid
4616   \bbl@exp{%
4617     \let\\\bbl@temp@pfam\<\bbl@stripslash#4\space>% eg, '\rmfamily '
4618     \<keys_if_exist:nnF>{fontspec-opentype}{Script/\bbl@cl{sname}}%
4619       {\\\newfontscript{\bbl@cl{sname}}{\bbl@cl{sotf}}}%
4620     \<keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@cl{lname}}%
4621       {\\\newfontlanguage{\bbl@cl{lname}}{\bbl@cl{lotf}}}%
4622     \let\\\bbl@tempfs@nx\<__fontspec_warning:nx>%
4623     \let\<__fontspec_warning:nx>\\\bbl@fs@warn@nx
4624     \let\\\bbl@tempfs@nxx\<__fontspec_warning:nxx>%
4625     \let\<__fontspec_warning:nxx>\\\bbl@fs@warn@nxx
4626     \\\renewfontfamily\\#4%
4627       [\bbl@cl{lsys},#2]}{#3}% ie \bbl@exp{..}{#3}
4628   \bbl@exp{%
4629     \let\<__fontspec_warning:nx>\\\bbl@tempfs@nx
4630     \let\<__fontspec_warning:nxx>\\\bbl@tempfs@nxx}%
4631   \begingroup
4632     #4%
4633     \xdef#1{\f@family}%     eg, \bbl@rmdflt@lang{FreeSerif(0)}
4634   \endgroup
4635   \let#4\bbl@temp@fam
4636   \bbl@exp{\let\<\bbl@stripslash#4\space>}\bbl@temp@pfam
4637   \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.

```
4638 \def\bbl@font@rst#1#2#3#4{%
4639   \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.

```
4640 \def\bbl@font@fams{rm,sf,tt}
4641 ⟨⟨/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.

```
4642 ⟨⟨*Footnote changes⟩⟩ ≡
```

```
4643 \bbl@trace{Bidi footnotes}
4644 \ifnum\bbl@bidimode>\z@
4645   \def\bbl@footnote#1#2#3{%
4646     \@ifnextchar[%
4647       {\bbl@footnote@o{#1}{#2}{#3}}%
4648       {\bbl@footnote@x{#1}{#2}{#3}}}
4649   \long\def\bbl@footnote@x#1#2#3#4{%
4650     \bgroup
4651       \select@language@x{\bbl@main@language}%
4652       \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
4653     \egroup}
4654   \long\def\bbl@footnote@o#1#2#3[#4]#5{%
4655     \bgroup
4656       \select@language@x{\bbl@main@language}%
4657       \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
4658     \egroup}
4659   \def\bbl@footnotetext#1#2#3{%
4660     \@ifnextchar[%
4661       {\bbl@footnotetext@o{#1}{#2}{#3}}%
4662       {\bbl@footnotetext@x{#1}{#2}{#3}}}
4663   \long\def\bbl@footnotetext@x#1#2#3#4{%
4664     \bgroup
4665       \select@language@x{\bbl@main@language}%
4666       \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
4667     \egroup}
4668   \long\def\bbl@footnotetext@o#1#2#3[#4]#5{%
4669     \bgroup
4670       \select@language@x{\bbl@main@language}%
4671       \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
4672     \egroup}
4673   \def\BabelFootnote#1#2#3#4{%
4674     \ifx\bbl@fn@footnote\@undefined
4675       \let\bbl@fn@footnote\footnote
4676     \fi
4677     \ifx\bbl@fn@footnotetext\@undefined
4678       \let\bbl@fn@footnotetext\footnotetext
4679     \fi
4680     \bbl@ifblank{#2}%
4681       {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
4682        \@namedef{\bbl@stripslash#1text}%
4683          {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
4684       {\def#1{\bbl@exp{\\\bbl@footnote{\\\foreignlanguage{#2}}}{#3}{#4}}%
4685        \@namedef{\bbl@stripslash#1text}%
4686          {\bbl@exp{\\\bbl@footnotetext{\\\foreignlanguage{#2}}}{#3}{#4}}}}
4687 \fi
4688 ⟨⟨/Footnote changes⟩⟩
```

Now, the code.

```
4689 ⟨∗xetex⟩
4690 \def\BabelStringsDefault{unicode}
4691 \let\xebbl@stop\relax
4692 \AddBabelHook{xetex}{encodedcommands}{%
4693   \def\bbl@tempa{#1}%
4694   \ifx\bbl@tempa\@empty
4695     \XeTeXinputencoding"bytes"%
4696   \else
4697     \XeTeXinputencoding"#1"%
4698   \fi
4699   \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4700 \AddBabelHook{xetex}{stopcommands}{%
4701   \xebbl@stop
4702   \let\xebbl@stop\relax}
4703 \def\bbl@intraspace#1 #2 #3\@@{%
```

159

```
4704  \bbl@csarg\gdef{xeisp@\languagename}%
4705    {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4706 \def\bbl@intrapenalty#1\@@{%
4707  \bbl@csarg\gdef{xeipn@\languagename}%
4708    {\XeTeXlinebreakpenalty #1\relax}}
4709 \def\bbl@provide@intraspace{%
4710  \bbl@xin@{/s}{/\bbl@cl{lnbrk}}%
4711  \ifin@\else\bbl@xin@{/c}{/\bbl@cl{lnbrk}}\fi
4712  \ifin@
4713    \bbl@ifunset{bbl@intsp@\languagename}{}%
4714      {\expandafter\ifx\csname bbl@intsp@\languagename\endcsname\@empty\else
4715        \ifx\bbl@KVP@intraspace\@nnil
4716          \bbl@exp{%
4717            \\\bbl@intraspace\bbl@cl{intsp}\\\@@}%
4718        \fi
4719        \ifx\bbl@KVP@intrapenalty\@nnil
4720          \bbl@intrapenalty0\@@
4721        \fi
4722      \fi
4723      \ifx\bbl@KVP@intraspace\@nnil\else % We may override the ini
4724        \expandafter\bbl@intraspace\bbl@KVP@intraspace\@@
4725      \fi
4726      \ifx\bbl@KVP@intrapenalty\@nnil\else
4727        \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
4728      \fi
4729      \bbl@exp{%
4730        % TODO. Execute only once (but redundant):
4731        \\\bbl@add\<extras\languagename>{%
4732          \XeTeXlinebreaklocale "\bbl@cl{tbcp}"%
4733          \<bbl@xeisp@\languagename>%
4734          \<bbl@xeipn@\languagename>}%
4735        \\\bbl@toglobal\<extras\languagename>%
4736        \\\bbl@add\<noextras\languagename>{%
4737          \XeTeXlinebreaklocale ""}%
4738        \\\bbl@toglobal\<noextras\languagename>}%
4739      \ifx\bbl@ispacesize\@undefined
4740        \gdef\bbl@ispacesize{\bbl@cl{xeisp}}%
4741        \ifx\AtBeginDocument\@notprerr
4742          \expandafter\@secondoftwo  % to execute right now
4743        \fi
4744        \AtBeginDocument{\bbl@patchfont{\bbl@ispacesize}}%
4745      \fi}%
4746  \fi}
4747 \ifx\DisableBabelHook\@undefined\endinput\fi
4748 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4749 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
4750 \DisableBabelHook{babel-fontspec}
4751 ⟨⟨Font selection⟩⟩
4752 \def\bbl@provide@extra#1{}
4753 ⟨/xetex⟩
```

12.2 Layout

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.

```
4754 ⟨∗xetex | texxet⟩
4755 \providecommand\bbl@provide@intraspace{}
4756 \bbl@trace{Redefinitions for bidi layout}
4757 \def\bbl@sspre@caption{%
```

```
4758    \bbl@exp{\everyhbox{\\\bbl@textdir\bbl@cs{wdir@\bbl@main@language}}}}
4759 \ifx\bbl@opt@layout\@nnil\else % if layout=..
4760 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
4761 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
4762 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
4763    \def\@hangfrom#1{%
4764       \setbox\@tempboxa\hbox{{#1}}%
4765       \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
4766       \noindent\box\@tempboxa}
4767    \def\raggedright{%
4768       \let\\\@centercr
4769       \bbl@startskip\z@skip
4770       \@rightskip\@flushglue
4771       \bbl@endskip\@rightskip
4772       \parindent\z@
4773       \parfillskip\bbl@startskip}
4774    \def\raggedleft{%
4775       \let\\\@centercr
4776       \bbl@startskip\@flushglue
4777       \bbl@endskip\z@skip
4778       \parindent\z@
4779       \parfillskip\bbl@endskip}
4780 \fi
4781 \IfBabelLayout{lists}
4782    {\bbl@sreplace\list
4783       {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
4784    \def\bbl@listleftmargin{%
4785       \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
4786    \ifcase\bbl@engine
4787       \def\labelenumii()\theenumii(}% pdftex doesn't reverse ()
4788       \def\p@enumiii{\p@enumii)\theenumii(}%
4789    \fi
4790    \bbl@sreplace\@verbatim
4791       {\leftskip\@totalleftmargin}%
4792       {\bbl@startskip\textwidth
4793          \advance\bbl@startskip-\linewidth}%
4794    \bbl@sreplace\@verbatim
4795       {\rightskip\z@skip}%
4796       {\bbl@endskip\z@skip}}%
4797    {}
4798 \IfBabelLayout{contents}
4799    {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
4800     \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
4801    {}
4802 \IfBabelLayout{columns}
4803    {\bbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbl@outputhbox}%
4804    \def\bbl@outputhbox#1{%
4805       \hb@xt@\textwidth{%
4806          \hskip\columnwidth
4807          \hfil
4808          {\normalcolor\vrule \@width\columnseprule}%
4809          \hfil
4810          \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
4811          \hskip-\textwidth
4812          \hb@xt@\columnwidth{\box\@outputbox \hss}%
4813          \hskip\columnsep
4814          \hskip\columnwidth}}}%
4815    {}
4816 ⟨⟨Footnote changes⟩⟩
4817 \IfBabelLayout{footnotes}%
4818    {\BabelFootnote\footnote\languagename{}{}%
4819     \BabelFootnote\localfootnote\languagename{}{}%
4820     \BabelFootnote\mainfootnote{}{}{}}
```

```
4821    {}
```

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.

```
4822 \IfBabelLayout{counters*}%
4823   {\bbl@add\bbl@opt@layout{.counters.}%
4824    \AddToHook{shipout/before}{%
4825      \let\bbl@tempa\babelsublr
4826      \let\babelsublr\@firstofone
4827      \let\bbl@save@thepage\thepage
4828      \protected@edef\thepage{\thepage}%
4829      \let\babelsublr\bbl@tempa}%
4830    \AddToHook{shipout/after}{%
4831      \let\thepage\bbl@save@thepage}}{}
4832 \IfBabelLayout{counters}%
4833   {\let\bbl@latinarabic=\@arabic
4834    \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
4835    \let\bbl@asciiroman=\@roman
4836    \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciiroman#1}}}%
4837    \let\bbl@asciiRoman=\@Roman
4838    \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}}{}
4839 \fi % end if layout
4840 ⟨/xetex | texxet⟩
```

12.3   8-bit TeX

Which start just above, because some code is shared with xetex. Now, 8-bit specific stuff.

```
4841 ⟨*texxet⟩
4842 \def\bbl@provide@extra#1{%
4843  % == auto-select encoding ==
4844  \ifx\bbl@encoding@select@off\@empty\else
4845    \bbl@ifunset{bbl@encoding@#1}%
4846      {\def\@elt##1{,##1,}%
4847       \edef\bbl@tempe{\expandafter\@gobbletwo\@fontenc@load@list}%
4848       \count@\z@
4849       \bbl@foreach\bbl@tempe{%
4850         \def\bbl@tempd{##1}%  Save last declared
4851         \advance\count@\@ne}%
4852       \ifnum\count@>\@ne
4853         \getlocaleproperty*\bbl@tempa{#1}{identification/encodings}%
4854         \ifx\bbl@tempa\relax \let\bbl@tempa\@empty \fi
4855         \bbl@replace\bbl@tempa{ }{,}%
4856         \global\bbl@csarg\let{encoding@#1}\@empty
4857         \bbl@xin@{,\bbl@tempd,}{,\bbl@tempa,}%
4858         \ifin@\else % if main encoding included in ini, do nothing
4859           \let\bbl@tempb\relax
4860           \bbl@foreach\bbl@tempa{%
4861             \ifx\bbl@tempb\relax
4862               \bbl@xin@{,##1,}{,\bbl@tempe,}%
4863               \ifin@\def\bbl@tempb{##1}\fi
4864             \fi}%
4865           \ifx\bbl@tempb\relax\else
4866             \bbl@exp{%
4867               \global\<bbl@add>\<bbl@preextras@#1>{\<bbl@encoding@#1>}%
4868               \gdef\<bbl@encoding@#1>{%
4869                 \\\babel@save\\\f@encoding
4870                 \\\bbl@add\\\originalTeX{\\\selectfont}%
4871                 \\\fontencoding{\bbl@tempb}%
4872                 \\\selectfont}}%
4873           \fi
4874         \fi
4875       \fi}%
4876      {}%
```

```
4877   \fi}
4878 ⟨/texxet⟩
```

12.4 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).

```
4879 ⟨*luatex⟩
4880 \ifx\AddBabelHook\@undefined % When plain.def, babel.sty starts
4881 \bbl@trace{Read language.dat}
4882 \ifx\bbl@readstream\@undefined
4883   \csname newread\endcsname\bbl@readstream
4884 \fi
4885 \begingroup
4886   \toks@{}
4887   \count@\z@ % 0=start, 1=0th, 2=normal
4888   \def\bbl@process@line#1#2 #3 #4 {%
4889     \ifx=#1%
4890       \bbl@process@synonym{#2}%
4891     \else
4892       \bbl@process@language{#1#2}{#3}{#4}%
4893     \fi
4894     \ignorespaces}
4895   \def\bbl@manylang{%
4896     \ifnum\bbl@last>\@ne
4897       \bbl@info{Non-standard hyphenation setup}%
4898     \fi
4899     \let\bbl@manylang\relax}
4900   \def\bbl@process@language#1#2#3{%
4901     \ifcase\count@
4902       \@ifundefined{zth@#1}{\count@\tw@}{\count@\@ne}%
4903     \or
4904       \count@\tw@
```

163

```
4905      \fi
4906      \ifnum\count@=\tw@
4907        \expandafter\addlanguage\csname l@#1\endcsname
4908        \language\allocationnumber
4909        \chardef\bbl@last\allocationnumber
4910        \bbl@manylang
4911        \let\bbl@elt\relax
4912        \xdef\bbl@languages{%
4913          \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
4914      \fi
4915      \the\toks@
4916      \toks@{}}
4917 \def\bbl@process@synonym@aux#1#2{%
4918      \global\expandafter\chardef\csname l@#1\endcsname#2\relax
4919      \let\bbl@elt\relax
4920      \xdef\bbl@languages{%
4921        \bbl@languages\bbl@elt{#1}{#2}{}{}}}%
4922 \def\bbl@process@synonym#1{%
4923      \ifcase\count@
4924        \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
4925      \or
4926        \@ifundefined{zth@#1}{\bbl@process@synonym@aux{#1}{0}}{}%
4927      \else
4928        \bbl@process@synonym@aux{#1}{\the\bbl@last}%
4929      \fi}
4930 \ifx\bbl@languages\@undefined % Just a (sensible?) guess
4931      \chardef\l@english\z@
4932      \chardef\l@USenglish\z@
4933      \chardef\bbl@last\z@
4934      \global\@namedef{bbl@hyphendata@0}{{hyphen.tex}{}}
4935      \gdef\bbl@languages{%
4936        \bbl@elt{english}{0}{hyphen.tex}{}%
4937        \bbl@elt{USenglish}{0}{}{}}
4938 \else
4939      \global\let\bbl@languages@format\bbl@languages
4940      \def\bbl@elt#1#2#3#4{% Remove all except language 0
4941        \ifnum#2>\z@\else
4942          \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
4943        \fi}%
4944      \xdef\bbl@languages{\bbl@languages}%
4945 \fi
4946 \def\bbl@elt#1#2#3#4{\@namedef{zth@#1}{}} % Define flags
4947 \bbl@languages
4948 \openin\bbl@readstream=language.dat
4949 \ifeof\bbl@readstream
4950      \bbl@warning{I couldn't find language.dat. No additional\\%
4951                   patterns loaded. Reported}%
4952 \else
4953      \loop
4954        \endlinechar\m@ne
4955        \read\bbl@readstream to \bbl@line
4956        \endlinechar`\^^M
4957        \if T\ifeof\bbl@readstream F\fi T\relax
4958          \ifx\bbl@line\@empty\else
4959            \edef\bbl@line{\bbl@line\space\space\space}%
4960            \expandafter\bbl@process@line\bbl@line\relax
4961          \fi
4962      \repeat
4963 \fi
4964 \closein\bbl@readstream
4965 \endgroup
4966 \bbl@trace{Macros for reading patterns files}
4967 \def\bbl@get@enc#1:#2:#3\@@@{\def\bbl@hyph@enc{#2}}
```

164

```
4968 \ifx\babelcatcodetablenum\@undefined
4969   \ifx\newcatcodetable\@undefined
4970     \def\babelcatcodetablenum{5211}
4971     \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4972   \else
4973     \newcatcodetable\babelcatcodetablenum
4974     \newcatcodetable\bbl@pattcodes
4975   \fi
4976 \else
4977   \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4978 \fi
4979 \def\bbl@luapatterns#1#2{%
4980   \bbl@get@enc#1::\@@@
4981   \setbox\z@\hbox\bgroup
4982     \begingroup
4983       \savecatcodetable\babelcatcodetablenum\relax
4984       \initcatcodetable\bbl@pattcodes\relax
4985       \catcodetable\bbl@pattcodes\relax
4986         \catcode`\#=6  \catcode`\$=3 \catcode`\&=4 \catcode`\^=7
4987         \catcode`\_=8  \catcode`\{=1 \catcode`\}=2 \catcode`\~=13
4988         \catcode`\@=11 \catcode`\^^I=10 \catcode`\^^J=12
4989         \catcode`\<=12 \catcode`\>=12 \catcode`\*=12 \catcode`\.=12
4990         \catcode`\-=12 \catcode`\/=12 \catcode`\[=12 \catcode`\]=12
4991         \catcode`\`=12 \catcode`\'=12 \catcode`\"=12
4992         \input #1\relax
4993       \catcodetable\babelcatcodetablenum\relax
4994     \endgroup
4995     \def\bbl@tempa{#2}%
4996     \ifx\bbl@tempa\@empty\else
4997       \input #2\relax
4998     \fi
4999   \egroup}%
5000 \def\bbl@patterns@lua#1{%
5001   \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
5002     \csname l@#1\endcsname
5003     \edef\bbl@tempa{#1}%
5004   \else
5005     \csname l@#1:\f@encoding\endcsname
5006     \edef\bbl@tempa{#1:\f@encoding}%
5007   \fi\relax
5008   \@namedef{lu@texhyphen@loaded@\the\language}{}% Temp
5009   \@ifundefined{bbl@hyphendata@\the\language}%
5010     {\def\bbl@elt##1##2##3##4{%
5011       \ifnum##2=\csname l@\bbl@tempa\endcsname % #2=spanish, dutch:OT1...
5012         \def\bbl@tempb{##3}%
5013         \ifx\bbl@tempb\@empty\else % if not a synonymous
5014           \def\bbl@tempc{{##3}{##4}}%
5015         \fi
5016         \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5017       \fi}%
5018     \bbl@languages
5019     \@ifundefined{bbl@hyphendata@\the\language}%
5020       {\bbl@info{No hyphenation patterns were set for\\%
5021                 language '\bbl@tempa'. Reported}}%
5022       {\expandafter\expandafter\expandafter\bbl@luapatterns
5023         \csname bbl@hyphendata@\the\language\endcsname}}{}}
5024 \endinput\fi
5025   % Here ends \ifx\AddBabelHook\@undefined
5026   % A few lines are only read by hyphen.cfg
5027 \ifx\DisableBabelHook\@undefined
5028   \AddBabelHook{luatex}{everylanguage}{%
5029     \def\process@language##1##2##3{%
5030       \def\process@line####1####2 ####3 ####4 {}}}
```

```
5031  \AddBabelHook{luatex}{loadpatterns}{%
5032    \input #1\relax
5033    \expandafter\gdef\csname bbl@hyphendata@\the\language\endcsname
5034      {{#1}{}}}
5035  \AddBabelHook{luatex}{loadexceptions}{%
5036    \input #1\relax
5037    \def\bbl@tempb##1##2{{##1}{#1}}%
5038    \expandafter\xdef\csname bbl@hyphendata@\the\language\endcsname
5039      {\expandafter\expandafter\expandafter\bbl@tempb
5040       \csname bbl@hyphendata@\the\language\endcsname}}
5041  \endinput\fi
5042  % Here stops reading code for hyphen.cfg
5043  % The following is read the 2nd time it's loaded
5044  \begingroup  % TODO - to a lua file
5045  \catcode`\%=12
5046  \catcode`\'=12
5047  \catcode`\"=12
5048  \catcode`\:=12
5049  \directlua{
5050    Babel = Babel or {}
5051    function Babel.bytes(line)
5052      return line:gsub("(.)",
5053        function (chr) return unicode.utf8.char(string.byte(chr)) end)
5054    end
5055    function Babel.begin_process_input()
5056      if luatexbase and luatexbase.add_to_callback then
5057        luatexbase.add_to_callback('process_input_buffer',
5058                                   Babel.bytes,'Babel.bytes')
5059      else
5060        Babel.callback = callback.find('process_input_buffer')
5061        callback.register('process_input_buffer',Babel.bytes)
5062      end
5063    end
5064    function Babel.end_process_input ()
5065      if luatexbase and luatexbase.remove_from_callback then
5066        luatexbase.remove_from_callback('process_input_buffer','Babel.bytes')
5067      else
5068        callback.register('process_input_buffer',Babel.callback)
5069      end
5070    end
5071    function Babel.addpatterns(pp, lg)
5072      local lg = lang.new(lg)
5073      local pats = lang.patterns(lg) or ''
5074      lang.clear_patterns(lg)
5075      for p in pp:gmatch('[^%s]+') do
5076        ss = ''
5077        for i in string.utfcharacters(p:gsub('%d', '')) do
5078          ss = ss .. '%d?' .. i
5079        end
5080        ss = ss:gsub('^%%d%?%.', '%%.') .. '%d?'
5081        ss = ss:gsub('%.%%d%?$', '%%.')
5082        pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
5083        if n == 0 then
5084          tex.sprint(
5085            [[\string\csname\space bbl@info\endcsname{New pattern: ]]
5086            .. p .. [[}]])
5087          pats = pats .. ' ' .. p
5088        else
5089          tex.sprint(
5090            [[\string\csname\space bbl@info\endcsname{Renew pattern: ]]
5091            .. p .. [[}]])
5092        end
5093      end
```

```
5094        lang.patterns(lg, pats)
5095      end
5096      Babel.characters = Babel.characters or {}
5097      Babel.ranges = Babel.ranges or {}
5098      function Babel.hlist_has_bidi(head)
5099        local has_bidi = false
5100        local ranges = Babel.ranges
5101        for item in node.traverse(head) do
5102          if item.id == node.id'glyph' then
5103            local itemchar = item.char
5104            local chardata = Babel.characters[itemchar]
5105            local dir = chardata and chardata.d or nil
5106            if not dir then
5107              for nn, et in ipairs(ranges) do
5108                if itemchar < et[1] then
5109                  break
5110                elseif itemchar <= et[2] then
5111                  dir = et[3]
5112                  break
5113                end
5114              end
5115            end
5116            if dir and (dir == 'al' or dir == 'r') then
5117              has_bidi = true
5118            end
5119          end
5120        end
5121        return has_bidi
5122      end
5123      function Babel.set_chranges_b (script, chrng)
5124        if chrng == '' then return end
5125        texio.write('Replacing ' .. script .. ' script ranges')
5126        Babel.script_blocks[script] = {}
5127        for s, e in string.gmatch(chrng..' ', '(.-)%.%.(.-)%s') do
5128          table.insert(
5129            Babel.script_blocks[script], {tonumber(s,16), tonumber(e,16)})
5130        end
5131      end
5132      function Babel.discard_sublr(str)
5133        if str:find( [[\string\indexentry]] ) and
5134            str:find( [[\string\babelsublr]] ) then
5135          str = str:gsub( [[\string\babelsublr%s*(%b{})]],
5136                          function(m) return m:sub(2,-2) end )
5137        end
5138        return str
5139 end
5140 }
5141 \endgroup
5142 \ifx\newattribute\@undefined\else
5143   \newattribute\bbl@attr@locale
5144   \directlua{ Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale' }
5145   \AddBabelHook{luatex}{beforeextras}{%
5146     \setattribute\bbl@attr@locale\localeid}
5147 \fi
5148 \def\BabelStringsDefault{unicode}
5149 \let\luabbl@stop\relax
5150 \AddBabelHook{luatex}{encodedcommands}{%
5151   \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
5152   \ifx\bbl@tempa\bbl@tempb\else
5153     \directlua{Babel.begin_process_input()}%
5154     \def\luabbl@stop{%
5155       \directlua{Babel.end_process_input()}}%
5156   \fi}%
```

```
5157 \AddBabelHook{luatex}{stopcommands}{%
5158   \luabbl@stop
5159   \let\luabbl@stop\relax}
5160 \AddBabelHook{luatex}{patterns}{%
5161   \@ifundefined{bbl@hyphendata@\the\language}%
5162     {\def\bbl@elt##1##2##3##4{%
5163        \ifnum##2=\csname l@#2\endcsname % #2=spanish, dutch:OT1...
5164          \def\bbl@tempb{##3}%
5165          \ifx\bbl@tempb\@empty\else % if not a synonymous
5166            \def\bbl@tempc{{##3}{##4}}%
5167          \fi
5168          \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5169        \fi}%
5170      \bbl@languages
5171      \@ifundefined{bbl@hyphendata@\the\language}%
5172        {\bbl@info{No hyphenation patterns were set for\\%
5173                  language '#2'. Reported}}%
5174        {\expandafter\expandafter\expandafter\bbl@luapatterns
5175          \csname bbl@hyphendata@\the\language\endcsname}}{}%
5176   \@ifundefined{bbl@patterns@}{}{%
5177     \begingroup
5178       \bbl@xin@{,\number\language,}{,\bbl@pttnlist}%
5179       \ifin@\else
5180         \ifx\bbl@patterns@\@empty\else
5181           \directlua{ Babel.addpatterns(
5182             [[\bbl@patterns@]], \number\language) }%
5183         \fi
5184         \@ifundefined{bbl@patterns@#1}%
5185           \@empty
5186           {\directlua{ Babel.addpatterns(
5187               [[\space\csname bbl@patterns@#1\endcsname]],
5188               \number\language) }}%
5189         \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
5190       \fi
5191     \endgroup}%
5192   \bbl@exp{%
5193     \bbl@ifunset{bbl@prehc@\languagename}{}%
5194       {\\\bbl@ifblank{\bbl@cs{prehc@\languagename}}{}%
5195         {\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.

```
5196 \@onlypreamble\babelpatterns
5197 \AtEndOfPackage{%
5198   \newcommand\babelpatterns[2][\@empty]{%
5199     \ifx\bbl@patterns@\relax
5200       \let\bbl@patterns@\@empty
5201     \fi
5202     \ifx\bbl@pttnlist\@empty\else
5203       \bbl@warning{%
5204         You must not intermingle \string\selectlanguage\space and\\%
5205         \string\babelpatterns\space or some patterns will not\\%
5206         be taken into account. Reported}%
5207     \fi
5208     \ifx\@empty#1%
5209       \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
5210     \else
5211       \edef\bbl@tempb{\zap@space#1 \@empty}%
5212       \bbl@for\bbl@tempa\bbl@tempb{%
5213         \bbl@fixname\bbl@tempa
5214         \bbl@iflanguage\bbl@tempa{%
5215           \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
```

```
5216          \@ifundefined{bbl@patterns@\bbl@tempa}%
5217            \@empty
5218            {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
5219          #2}}}%
5220    \fi}}
```

12.5   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.

```
5221 % TODO - to a lua file
5222 \directlua{
5223   Babel = Babel or {}
5224   Babel.linebreaking = Babel.linebreaking or {}
5225   Babel.linebreaking.before = {}
5226   Babel.linebreaking.after = {}
5227   Babel.locale = {} % Free to use, indexed by \localeid
5228   function Babel.linebreaking.add_before(func, pos)
5229     tex.print([[\noexpand\csname bbl@luahyphenate\endcsname]])
5230     if pos == nil then
5231       table.insert(Babel.linebreaking.before, func)
5232     else
5233       table.insert(Babel.linebreaking.before, pos, func)
5234     end
5235   end
5236   function Babel.linebreaking.add_after(func)
5237     tex.print([[\noexpand\csname bbl@luahyphenate\endcsname]])
5238     table.insert(Babel.linebreaking.after, func)
5239   end
5240 }
5241 \def\bbl@intraspace#1 #2 #3\@@{%
5242   \directlua{
5243     Babel = Babel or {}
5244     Babel.intraspaces = Babel.intraspaces or {}
5245     Babel.intraspaces['\csname bbl@sbcp@\languagename\endcsname'] = %
5246       {b = #1, p = #2, m = #3}
5247     Babel.locale_props[\the\localeid].intraspace = %
5248       {b = #1, p = #2, m = #3}
5249   }}
5250 \def\bbl@intrapenalty#1\@@{%
5251   \directlua{
5252     Babel = Babel or {}
5253     Babel.intrapenalties = Babel.intrapenalties or {}
5254     Babel.intrapenalties['\csname bbl@sbcp@\languagename\endcsname'] = #1
5255     Babel.locale_props[\the\localeid].intrapenalty = #1
5256   }}
5257 \begingroup
5258 \catcode`\%=12
5259 \catcode`\^=14
5260 \catcode`\'=12
5261 \catcode`\~=12
5262 \gdef\bbl@seaintraspace{^
5263   \let\bbl@seaintraspace\relax
5264   \directlua{
5265     Babel = Babel or {}
5266     Babel.sea_enabled = true
5267     Babel.sea_ranges = Babel.sea_ranges or {}
5268     function Babel.set_chranges (script, chrng)
5269       local c = 0
5270       for s, e in string.gmatch(chrng..' ', '(.-)%.%.(.-)%s') do
5271         Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
```

```
5272         c = c + 1
5273       end
5274     end
5275     function Babel.sea_disc_to_space (head)
5276       local sea_ranges = Babel.sea_ranges
5277       local last_char = nil
5278       local quad = 655360      ^% 10 pt = 655360 = 10 * 65536
5279       for item in node.traverse(head) do
5280         local i = item.id
5281         if i == node.id'glyph' then
5282           last_char = item
5283         elseif i == 7 and item.subtype == 3 and last_char
5284             and last_char.char > 0x0C99 then
5285           quad = font.getfont(last_char.font).size
5286           for lg, rg in pairs(sea_ranges) do
5287             if last_char.char > rg[1] and last_char.char < rg[2] then
5288               lg = lg:sub(1, 4)  ^% Remove trailing number of, eg, Cyrl1
5289               local intraspace = Babel.intraspaces[lg]
5290               local intrapenalty = Babel.intrapenalties[lg]
5291               local n
5292               if intrapenalty ~= 0 then
5293                 n = node.new(14, 0)      ^% penalty
5294                 n.penalty = intrapenalty
5295                 node.insert_before(head, item, n)
5296               end
5297               n = node.new(12, 13)      ^% (glue, spaceskip)
5298               node.setglue(n, intraspace.b * quad,
5299                               intraspace.p * quad,
5300                               intraspace.m * quad)
5301               node.insert_before(head, item, n)
5302               node.remove(head, item)
5303             end
5304           end
5305         end
5306       end
5307     end
5308 }^^
5309 \bbl@luahyphenate}
```

12.6   CJK line breaking

Minimal line breaking for CJK scripts, mainly intended for simple documents and short texts as a secondary language. Only line breaking, with a little stretching for justification, without any attempt to adjust the spacing. It is based on (but does not strictly follow) the Unicode algorithm.

We first need a little table with the corresponding line breaking properties. A few characters have an additional key for the width (fullwidth *vs.* halfwidth), not yet used. There is a separate file, defined below.

```
5310 \catcode`\%=14
5311 \gdef\bbl@cjkintraspace{%
5312   \let\bbl@cjkintraspace\relax
5313   \directlua{
5314     Babel = Babel or {}
5315     require('babel-data-cjk.lua')
5316     Babel.cjk_enabled = true
5317     function Babel.cjk_linebreak(head)
5318       local GLYPH = node.id'glyph'
5319       local last_char = nil
5320       local quad = 655360      % 10 pt = 655360 = 10 * 65536
5321       local last_class = nil
5322       local last_lang = nil
5323
5324       for item in node.traverse(head) do
5325         if item.id == GLYPH then
```

```
5326
5327            local lang = item.lang
5328
5329            local LOCALE = node.get_attribute(item,
5330                 Babel.attr_locale)
5331            local props = Babel.locale_props[LOCALE]
5332
5333            local class = Babel.cjk_class[item.char].c
5334
5335            if props.cjk_quotes and props.cjk_quotes[item.char] then
5336              class = props.cjk_quotes[item.char]
5337            end
5338
5339            if class == 'cp' then class = 'cl' end % )] as CL
5340            if class == 'id' then class = 'I' end
5341
5342            local br = 0
5343            if class and last_class and Babel.cjk_breaks[last_class][class] then
5344              br = Babel.cjk_breaks[last_class][class]
5345            end
5346
5347            if br == 1 and props.linebreak == 'c' and
5348                lang ~= \the\l@nohyphenation\space and
5349                last_lang ~= \the\l@nohyphenation then
5350              local intrapenalty = props.intrapenalty
5351              if intrapenalty ~= 0 then
5352                local n = node.new(14, 0)      % penalty
5353                n.penalty = intrapenalty
5354                node.insert_before(head, item, n)
5355              end
5356              local intraspace = props.intraspace
5357              local n = node.new(12, 13)       % (glue, spaceskip)
5358              node.setglue(n, intraspace.b * quad,
5359                              intraspace.p * quad,
5360                              intraspace.m * quad)
5361              node.insert_before(head, item, n)
5362            end
5363
5364            if font.getfont(item.font) then
5365              quad = font.getfont(item.font).size
5366            end
5367            last_class = class
5368            last_lang = lang
5369          else % if penalty, glue or anything else
5370            last_class = nil
5371          end
5372        end
5373      lang.hyphenate(head)
5374    end
5375  }%
5376  \bbl@luahyphenate}
5377 \gdef\bbl@luahyphenate{%
5378  \let\bbl@luahyphenate\relax
5379  \directlua{
5380    luatexbase.add_to_callback('hyphenate',
5381    function (head, tail)
5382      if Babel.linebreaking.before then
5383        for k, func in ipairs(Babel.linebreaking.before)  do
5384          func(head)
5385        end
5386      end
5387      if Babel.cjk_enabled then
5388        Babel.cjk_linebreak(head)
```

```
5389        end
5390        lang.hyphenate(head)
5391        if Babel.linebreaking.after then
5392          for k, func in ipairs(Babel.linebreaking.after)  do
5393            func(head)
5394          end
5395        end
5396        if Babel.sea_enabled then
5397          Babel.sea_disc_to_space(head)
5398        end
5399      end,
5400      'Babel.hyphenate')
5401  }
5402 }
5403 \endgroup
5404 \def\bbl@provide@intraspace{%
5405   \bbl@ifunset{bbl@intsp@\languagename}{}%
5406     {\expandafter\ifx\csname bbl@intsp@\languagename\endcsname\@empty\else
5407      \bbl@xin@{/c}{/\bbl@cl{lnbrk}}%
5408      \ifin@          % cjk
5409        \bbl@cjkintraspace
5410        \directlua{
5411            Babel = Babel or {}
5412            Babel.locale_props = Babel.locale_props or {}
5413            Babel.locale_props[\the\localeid].linebreak = 'c'
5414        }%
5415        \bbl@exp{\\\bbl@intraspace\bbl@cl{intsp}\\\@@}%
5416        \ifx\bbl@KVP@intrapenalty\@nnil
5417          \bbl@intrapenalty0\@@
5418        \fi
5419      \else            % sea
5420        \bbl@seaintraspace
5421        \bbl@exp{\\\bbl@intraspace\bbl@cl{intsp}\\\@@}%
5422        \directlua{
5423            Babel = Babel or {}
5424            Babel.sea_ranges = Babel.sea_ranges or {}
5425            Babel.set_chranges('\bbl@cl{sbcp}',
5426                               '\bbl@cl{chrng}')
5427        }%
5428        \ifx\bbl@KVP@intrapenalty\@nnil
5429          \bbl@intrapenalty0\@@
5430        \fi
5431      \fi
5432    \fi
5433    \ifx\bbl@KVP@intrapenalty\@nnil\else
5434      \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
5435    \fi}}
```

12.7   Arabic justification

```
5436 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5437 \def\bblar@chars{%
5438  0628,0629,062A,062B,062C,062D,062E,062F,0630,0631,0632,0633,%
5439  0634,0635,0636,0637,0638,0639,063A,063B,063C,063D,063E,063F,%
5440  0640,0641,0642,0643,0644,0645,0646,0647,0649}
5441 \def\bblar@elongated{%
5442  0626,0628,062A,062B,0633,0634,0635,0636,063B,%
5443  063C,063D,063E,063F,0641,0642,0643,0644,0646,%
5444  0649,064A}
5445 \begingroup
5446   \catcode`\_=11 \catcode`\:=11
5447   \gdef\bblar@nofswarn{\gdef\msg_warning:nnx##1##2##3{}}
5448 \endgroup
```

```
5449 \gdef\bbl@arabicjust{%
5450   \let\bbl@arabicjust\relax
5451   \newattribute\bblar@kashida
5452   \directlua{ Babel.attr_kashida = luatexbase.registernumber'bblar@kashida' }%
5453   \bblar@kashida=\z@
5454   \bbl@patchfont{{\bbl@parsejalt}}%
5455   \directlua{
5456     Babel.arabic.elong_map   = Babel.arabic.elong_map or {}
5457     Babel.arabic.elong_map[\the\localeid]   = {}
5458     luatexbase.add_to_callback('post_linebreak_filter',
5459       Babel.arabic.justify, 'Babel.arabic.justify')
5460     luatexbase.add_to_callback('hpack_filter',
5461       Babel.arabic.justify_hbox, 'Babel.arabic.justify_hbox')
5462 }}%
5463 % Save both node lists to make replacement. TODO. Save also widths to
5464 % make computations
5465 \def\bblar@fetchjalt#1#2#3#4{%
5466   \bbl@exp{\\\bbl@foreach{#1}}{%
5467     \bbl@ifunset{bblar@JE@##1}%
5468       {\setbox\z@\hbox{^^^^200d\char"##1#2}}%
5469       {\setbox\z@\hbox{^^^^200d\char"\@nameuse{bblar@JE@##1}#2}}%
5470     \directlua{%
5471       local last = nil
5472       for item in node.traverse(tex.box[0].head) do
5473         if item.id == node.id'glyph' and item.char > 0x600 and
5474            not (item.char == 0x200D) then
5475           last = item
5476         end
5477       end
5478       Babel.arabic.#3['##1#4'] = last.char
5479     }}}
5480 % Brute force. No rules at all, yet. The ideal: look at jalt table. And
5481 % perhaps other tables (falt?, cswh?). What about kaf? And diacritic
5482 % positioning?
5483 \gdef\bbl@parsejalt{%
5484   \ifx\addfontfeature\@undefined\else
5485     \bbl@xin@{/e}{/\bbl@cl{lnbrk}}%
5486     \ifin@
5487       \directlua{%
5488         if Babel.arabic.elong_map[\the\localeid][\fontid\font] == nil then
5489           Babel.arabic.elong_map[\the\localeid][\fontid\font] = {}
5490           tex.print([[\string\csname\space bbl@parsejalti\endcsname]])
5491         end
5492       }%
5493     \fi
5494   \fi}
5495 \gdef\bbl@parsejalti{%
5496   \begingroup
5497     \let\bbl@parsejalt\relax     % To avoid infinite loop
5498     \edef\bbl@tempb{\fontid\font}%
5499     \bblar@nofswarn
5500     \bblar@fetchjalt\bblar@elongated{}{from}{}%
5501     \bblar@fetchjalt\bblar@chars{^^^^064a}{from}{a}% Alef maksura
5502     \bblar@fetchjalt\bblar@chars{^^^^0649}{from}{y}% Yeh
5503     \addfontfeature{RawFeature=+jalt}%
5504     % \@namedef{bblar@JE@0643}{06AA}% todo: catch medial kaf
5505     \bblar@fetchjalt\bblar@elongated{}{dest}{}%
5506     \bblar@fetchjalt\bblar@chars{^^^^064a}{dest}{a}%
5507     \bblar@fetchjalt\bblar@chars{^^^^0649}{dest}{y}%
5508       \directlua{%
5509         for k, v in pairs(Babel.arabic.from) do
5510           if Babel.arabic.dest[k] and
5511             not (Babel.arabic.from[k] == Babel.arabic.dest[k]) then
```

```
5512          Babel.arabic.elong_map[\the\localeid][\bbl@tempb]
5513              [Babel.arabic.from[k]] = Babel.arabic.dest[k]
5514        end
5515      end
5516    }%
5517  \endgroup}
5518 %
5519 \begingroup
5520 \catcode`#=11
5521 \catcode`~=11
5522 \directlua{
5523
5524 Babel.arabic = Babel.arabic or {}
5525 Babel.arabic.from = {}
5526 Babel.arabic.dest = {}
5527 Babel.arabic.justify_factor = 0.95
5528 Babel.arabic.justify_enabled = true
5529
5530 function Babel.arabic.justify(head)
5531   if not Babel.arabic.justify_enabled then return head end
5532   for line in node.traverse_id(node.id'hlist', head) do
5533     Babel.arabic.justify_hlist(head, line)
5534   end
5535   return head
5536 end
5537
5538 function Babel.arabic.justify_hbox(head, gc, size, pack)
5539   local has_inf = false
5540   if Babel.arabic.justify_enabled and pack == 'exactly' then
5541     for n in node.traverse_id(12, head) do
5542       if n.stretch_order > 0 then has_inf = true end
5543     end
5544     if not has_inf then
5545       Babel.arabic.justify_hlist(head, nil, gc, size, pack)
5546     end
5547   end
5548   return head
5549 end
5550
5551 function Babel.arabic.justify_hlist(head, line, gc, size, pack)
5552   local d, new
5553   local k_list, k_item, pos_inline
5554   local width, width_new, full, k_curr, wt_pos, goal, shift
5555   local subst_done = false
5556   local elong_map = Babel.arabic.elong_map
5557   local last_line
5558   local GLYPH = node.id'glyph'
5559   local KASHIDA = Babel.attr_kashida
5560   local LOCALE = Babel.attr_locale
5561
5562   if line == nil then
5563     line = {}
5564     line.glue_sign = 1
5565     line.glue_order = 0
5566     line.head = head
5567     line.shift = 0
5568     line.width = size
5569   end
5570
5571   % Exclude last line. todo. But-- it discards one-word lines, too!
5572   % ? Look for glue = 12:15
5573   if (line.glue_sign == 1 and line.glue_order == 0) then
5574     elongs = {}      % Stores elongated candidates of each line
```

174

```
5575    k_list = {}      % And all letters with kashida
5576    pos_inline = 0  % Not yet used
5577
5578    for n in node.traverse_id(GLYPH, line.head) do
5579      pos_inline = pos_inline + 1 % To find where it is. Not used.
5580
5581      % Elongated glyphs
5582      if elong_map then
5583        local locale = node.get_attribute(n, LOCALE)
5584        if elong_map[locale] and elong_map[locale][n.font] and
5585            elong_map[locale][n.font][n.char] then
5586          table.insert(elongs, {node = n, locale = locale} )
5587          node.set_attribute(n.prev, KASHIDA, 0)
5588        end
5589      end
5590
5591      % Tatwil
5592      if Babel.kashida_wts then
5593        local k_wt = node.get_attribute(n, KASHIDA)
5594        if k_wt > 0 then % todo. parameter for multi inserts
5595          table.insert(k_list, {node = n, weight = k_wt, pos = pos_inline})
5596        end
5597      end
5598
5599    end % of node.traverse_id
5600
5601    if #elongs == 0 and #k_list == 0 then goto next_line end
5602    full  = line.width
5603    shift = line.shift
5604    goal  = full * Babel.arabic.justify_factor % A bit crude
5605    width = node.dimensions(line.head)    % The 'natural' width
5606
5607    % == Elongated ==
5608    % Original idea taken from 'chikenize'
5609    while (#elongs > 0 and width < goal) do
5610      subst_done = true
5611      local x = #elongs
5612      local curr = elongs[x].node
5613      local oldchar = curr.char
5614      curr.char = elong_map[elongs[x].locale][curr.font][curr.char]
5615      width = node.dimensions(line.head)  % Check if the line is too wide
5616      % Substitute back if the line would be too wide and break:
5617      if width > goal then
5618        curr.char = oldchar
5619        break
5620      end
5621      % If continue, pop the just substituted node from the list:
5622      table.remove(elongs, x)
5623    end
5624
5625    % == Tatwil ==
5626    if #k_list == 0 then goto next_line end
5627
5628    width = node.dimensions(line.head)     % The 'natural' width
5629    k_curr = #k_list
5630    wt_pos = 1
5631
5632    while width < goal do
5633      subst_done = true
5634      k_item = k_list[k_curr].node
5635      if k_list[k_curr].weight == Babel.kashida_wts[wt_pos] then
5636        d = node.copy(k_item)
5637        d.char = 0x0640
```

```
5638          line.head, new = node.insert_after(line.head, k_item, d)
5639          width_new = node.dimensions(line.head)
5640          if width > goal or width == width_new then
5641            node.remove(line.head, new) % Better compute before
5642            break
5643          end
5644          width = width_new
5645        end
5646        if k_curr == 1 then
5647          k_curr = #k_list
5648          wt_pos = (wt_pos >= table.getn(Babel.kashida_wts)) and 1 or wt_pos+1
5649        else
5650          k_curr = k_curr - 1
5651        end
5652      end
5653
5654      ::next_line::
5655
5656      % Must take into account marks and ins, see luatex manual.
5657      % Have to be executed only if there are changes. Investigate
5658      % what's going on exactly.
5659      if subst_done and not gc then
5660        d = node.hpack(line.head, full, 'exactly')
5661        d.shift = shift
5662        node.insert_before(head, line, d)
5663        node.remove(head, line)
5664      end
5665   end % if process line
5666 end
5667 }
5668 \endgroup
5669 \fi\fi % Arabic just block
```

12.8 Common stuff

```
5670 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
5671 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
5672 \DisableBabelHook{babel-fontspec}
5673 ⟨⟨Font selection⟩⟩
```

12.9 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.

```
5674 % TODO - to a lua file
5675 \directlua{
5676 Babel.script_blocks = {
5677   ['dflt'] = {},
5678   ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
5679                {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
5680   ['Armn'] = {{0x0530, 0x058F}},
5681   ['Beng'] = {{0x0980, 0x09FF}},
5682   ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
5683   ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
5684   ['Cyrl'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
5685                {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
5686   ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
5687   ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
5688                {0xAB00, 0xAB2F}},
```

```
5689    ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
5690    % Don't follow strictly Unicode, which places some Coptic letters in
5691    % the 'Greek and Coptic' block
5692    ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
5693    ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
5694                {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
5695                {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
5696                {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
5697                {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
5698                {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
5699    ['Hebr'] = {{0x0590, 0x05FF}},
5700    ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
5701                {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
5702    ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
5703    ['Knda'] = {{0x0C80, 0x0CFF}},
5704    ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
5705                {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
5706                {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
5707    ['Laoo'] = {{0x0E80, 0x0EFF}},
5708    ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
5709                {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
5710                {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
5711    ['Mahj'] = {{0x11150, 0x1117F}},
5712    ['Mlym'] = {{0x0D00, 0x0D7F}},
5713    ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
5714    ['Orya'] = {{0x0B00, 0x0B7F}},
5715    ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
5716    ['Syrc'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
5717    ['Taml'] = {{0x0B80, 0x0BFF}},
5718    ['Telu'] = {{0x0C00, 0x0C7F}},
5719    ['Tfng'] = {{0x2D30, 0x2D7F}},
5720    ['Thai'] = {{0x0E00, 0x0E7F}},
5721    ['Tibt'] = {{0x0F00, 0x0FFF}},
5722    ['Vaii'] = {{0xA500, 0xA63F}},
5723    ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
5724 }
5725
5726 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
5727 Babel.script_blocks.Hant = Babel.script_blocks.Hans
5728 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
5729
5730 function Babel.locale_map(head)
5731   if not Babel.locale_mapped then return head end
5732
5733   local LOCALE = Babel.attr_locale
5734   local GLYPH = node.id('glyph')
5735   local inmath = false
5736   local toloc_save
5737   for item in node.traverse(head) do
5738     local toloc
5739     if not inmath and item.id == GLYPH then
5740       % Optimization: build a table with the chars found
5741       if Babel.chr_to_loc[item.char] then
5742         toloc = Babel.chr_to_loc[item.char]
5743       else
5744         for lc, maps in pairs(Babel.loc_to_scr) do
5745           for _, rg in pairs(maps) do
5746             if item.char >= rg[1] and item.char <= rg[2] then
5747               Babel.chr_to_loc[item.char] = lc
5748               toloc = lc
5749               break
5750             end
5751           end
```

```
5752            end
5753          end
5754          % Now, take action, but treat composite chars in a different
5755          % fashion, because they 'inherit' the previous locale. Not yet
5756          % optimized.
5757          if not toloc and
5758              (item.char >= 0x0300 and item.char <= 0x036F) or
5759              (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
5760              (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
5761            toloc = toloc_save
5762          end
5763          if toloc and Babel.locale_props[toloc] and
5764              Babel.locale_props[toloc].letters and
5765              tex.getcatcode(item.char) \string~= 11 then
5766            toloc = nil
5767          end
5768          if toloc and toloc > -1 then
5769            if Babel.locale_props[toloc].lg then
5770              item.lang = Babel.locale_props[toloc].lg
5771              node.set_attribute(item, LOCALE, toloc)
5772            end
5773            if Babel.locale_props[toloc]['/'..item.font] then
5774              item.font = Babel.locale_props[toloc]['/'..item.font]
5775            end
5776            toloc_save = toloc
5777          end
5778        elseif not inmath and item.id == 7 then % Apply recursively
5779          item.replace = item.replace and Babel.locale_map(item.replace)
5780          item.pre     = item.pre and Babel.locale_map(item.pre)
5781          item.post    = item.post and Babel.locale_map(item.post)
5782        elseif item.id == node.id'math' then
5783          inmath = (item.subtype == 0)
5784        end
5785      end
5786      return head
5787    end
5788  }
```

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

```
5789 \newcommand\babelcharproperty[1]{%
5790   \count@=#1\relax
5791   \ifvmode
5792     \expandafter\bbl@chprop
5793   \else
5794     \bbl@error{\string\babelcharproperty\space can be used only in\\%
5795               vertical mode (preamble or between paragraphs)}%
5796               {See the manual for futher info}%
5797   \fi}
5798 \newcommand\bbl@chprop[3][\the\count@]{%
5799   \@tempcnta=#1\relax
5800   \bbl@ifunset{bbl@chprop@#2}%
5801     {\bbl@error{No property named '#2'. Allowed values are\\%
5802               direction (bc), mirror (bmg), and linebreak (lb)}%
5803               {See the manual for futher info}}%
5804     {}%
5805   \loop
5806     \bbl@cs{chprop@#2}{#3}%
5807   \ifnum\count@<\@tempcnta
5808     \advance\count@\@ne
5809   \repeat}
5810 \def\bbl@chprop@direction#1{%
5811   \directlua{
```

```
5812      Babel.characters[\the\count@] =  Babel.characters[\the\count@] or {}
5813      Babel.characters[\the\count@]['d'] = '#1'
5814  }}
5815 \let\bbl@chprop@bc\bbl@chprop@direction
5816 \def\bbl@chprop@mirror#1{%
5817   \directlua{
5818      Babel.characters[\the\count@] =  Babel.characters[\the\count@] or {}
5819      Babel.characters[\the\count@]['m'] = '\number#1'
5820  }}
5821 \let\bbl@chprop@bmg\bbl@chprop@mirror
5822 \def\bbl@chprop@linebreak#1{%
5823   \directlua{
5824      Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
5825      Babel.cjk_characters[\the\count@]['c'] = '#1'
5826  }}
5827 \let\bbl@chprop@lb\bbl@chprop@linebreak
5828 \def\bbl@chprop@locale#1{%
5829   \directlua{
5830      Babel.chr_to_loc = Babel.chr_to_loc or {}
5831      Babel.chr_to_loc[\the\count@] =
5832        \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@@#1}}\space
5833  }}
```

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.

```
5834 \directlua{
5835   Babel.nohyphenation = \the\l@nohyphenation
5836 }
```

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).

```
5837 \begingroup
5838 \catcode`\~=12
5839 \catcode`\%=12
5840 \catcode`\&=14
5841 \catcode`\|=12
5842 \gdef\babelprehyphenation{&%
5843   \@ifnextchar[{\bbl@settransform{0}}{\bbl@settransform{0}[]}}
5844 \gdef\babelposthyphenation{&%
5845   \@ifnextchar[{\bbl@settransform{1}}{\bbl@settransform{1}[]}}
5846 \gdef\bbl@postlinebreak{\bbl@settransform{2}[]} &% WIP
5847 \gdef\bbl@settransform#1[#2]#3#4#5{&%
5848   \ifcase#1
5849     \bbl@activateprehyphen
5850   \or
5851     \bbl@activateposthyphen
5852   \fi
5853   \begingroup
5854     \def\babeltempa{\bbl@add@list\babeltempb}&%
5855     \let\babeltempb\@empty
5856     \def\bbl@tempa{#5}&%
5857     \bbl@replace\bbl@tempa{,}{ ,}&% TODO. Ugly trick to preserve {}
5858     \expandafter\bbl@foreach\expandafter{\bbl@tempa}{&%
5859       \bbl@ifsamestring{##1}{remove}&%
5860         {\bbl@add@list\babeltempb{nil}}&%
5861         {\directlua{
5862           local rep = [=[##1]=]
```

```
5863              rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
5864              rep = rep:gsub('^%s*(insert)%s*,', 'insert = true, ')
5865              rep = rep:gsub('(string)%s*=%s*([^%s,]*)', Babel.capture_func)
5866              if #1 == 0 or #1 == 2 then
5867                rep = rep:gsub('(space)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
5868                  'space = {' .. '%2, %3, %4' .. '}')
5869                rep = rep:gsub('(spacefactor)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
5870                  'spacefactor = {' .. '%2, %3, %4' .. '}')
5871                rep = rep:gsub('(kashida)%s*=%s*([^%s,]*)', Babel.capture_kashida)
5872              else
5873                rep = rep:gsub(    '(no)%s*=%s*([^%s,]*)', Babel.capture_func)
5874                rep = rep:gsub(   '(pre)%s*=%s*([^%s,]*)', Babel.capture_func)
5875                rep = rep:gsub(  '(post)%s*=%s*([^%s,]*)', Babel.capture_func)
5876              end
5877              tex.print([[\string\babeltempa{{]] .. rep .. [[}}]])
5878          }}}&%
5879      \bbl@foreach\babeltempb{&%
5880        \bbl@forkv{{##1}}{&%
5881          \in@{,####1,}{,nil,step,data,remove,insert,string,no,pre,&%
5882            no,post,penalty,kashida,space,spacefactor,}&%
5883          \ifin@\else
5884            \bbl@error
5885            {Bad option '####1' in a transform.\\&%
5886             I'll ignore it but expect more errors}&%
5887            {See the manual for further info.}&%
5888          \fi}}&%
5889      \let\bbl@kv@attribute\relax
5890      \let\bbl@kv@label\relax
5891      \let\bbl@kv@fonts\@empty
5892      \bbl@forkv{#2}{\bbl@csarg\edef{kv@##1}{##2}}&%
5893      \ifx\bbl@kv@fonts\@empty\else\bbl@settransfont\fi
5894      \ifx\bbl@kv@attribute\relax
5895        \ifx\bbl@kv@label\relax\else
5896          \bbl@exp{\\\bbl@trim@def\\\bbl@kv@fonts{\bbl@kv@fonts}}&%
5897          \bbl@replace\bbl@kv@fonts{ }{,}&%
5898          \edef\bbl@kv@attribute{bbl@ATR@\bbl@kv@label @#3@\bbl@kv@fonts}&%
5899          \count@\z@
5900          \def\bbl@elt##1##2##3{&%
5901            \bbl@ifsamestring{#3,\bbl@kv@label}{##1,##2}&%
5902              {\bbl@ifsamestring{\bbl@kv@fonts}{##3}&%
5903                {\count@\@ne}&%
5904                {\bbl@error
5905                  {Transforms cannot be re-assigned to different\\&%
5906                   fonts. The conflict is in '\bbl@kv@label'.\\&%
5907                   Apply the same fonts or use a different label}&%
5908                  {See the manual for further details.}}}&%
5909            {}}&%
5910          \bbl@transfont@list
5911          \ifnum\count@=\z@
5912            \bbl@exp{\global\\\bbl@add\\\bbl@transfont@list
5913              {\\\bbl@elt{#3}{\bbl@kv@label}{\bbl@kv@fonts}}}&%
5914          \fi
5915          \bbl@ifunset{\bbl@kv@attribute}&%
5916            {\global\bbl@carg\newattribute{\bbl@kv@attribute}}&%
5917            {}&%
5918          \global\bbl@carg\setattribute{\bbl@kv@attribute}\@ne
5919        \fi
5920      \else
5921        \edef\bbl@kv@attribute{\expandafter\bbl@stripslash\bbl@kv@attribute}&%
5922      \fi
5923      \directlua{
5924        local lbkr = Babel.linebreaking.replacements[#1]
5925        local u = unicode.utf8
```

```
5926      local id, attr, label
5927      if #1 == 0 or #1 == 2 then
5928        id = \the\csname bbl@id@@#3\endcsname\space
5929      else
5930        id = \the\csname l@#3\endcsname\space
5931      end
5932      \ifx\bbl@kv@attribute\relax
5933        attr = -1
5934      \else
5935        attr = luatexbase.registernumber'\bbl@kv@attribute'
5936      \fi
5937      \ifx\bbl@kv@label\relax\else  &% Same refs:
5938        label = [==[\bbl@kv@label]==]
5939      \fi
5940      &% Convert pattern:
5941      local patt = string.gsub([==[#4]==], '%s', '')
5942      if #1 == 0 or #1 == 2 then
5943        patt = string.gsub(patt, '|', ' ')
5944      end
5945      if not u.find(patt, '()', nil, true) then
5946        patt = '()' .. patt .. '()'
5947      end
5948      if #1 == 1 then
5949        patt = string.gsub(patt, '%(%)%^', '^()')
5950        patt = string.gsub(patt, '%$%(%)', '()$')
5951      end
5952      patt = u.gsub(patt, '{(.)}',
5953            function (n)
5954              return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
5955            end)
5956      patt = u.gsub(patt, '{(%x%x%x%x+)}',
5957            function (n)
5958              return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%1')
5959            end)
5960      lbkr[id] = lbkr[id] or {}
5961      table.insert(lbkr[id],
5962        { label=label, attr=attr, pattern=patt, replace={\babeltempb} })
5963    }&%
5964  \endgroup}
5965 \endgroup
5966 \let\bbl@transfont@list\@empty
5967 \def\bbl@settransfont{%
5968  \global\let\bbl@settransfont\relax % Execute only once
5969  \gdef\bbl@transfont{%
5970    \def\bbl@elt####1####2####3{%
5971      \bbl@ifblank{####3}%
5972        {\count@\tw@}% Do nothing if no fonts
5973        {\count@\z@
5974         \bbl@vforeach{####3}{%
5975           \def\bbl@tempd{########1}%
5976           \edef\bbl@tempe{\bbl@transfam/\f@series/\f@shape}%
5977           \ifx\bbl@tempd\bbl@tempe
5978             \count@\@ne
5979           \else\ifx\bbl@tempd\bbl@transfam
5980             \count@\@ne
5981           \fi\fi}%
5982         \ifcase\count@
5983           \bbl@csarg\unsetattribute{ATR@####2@####1@####3}%
5984         \or
5985           \bbl@csarg\setattribute{ATR@####2@####1@####3}\@ne
5986         \fi}}%
5987      \bbl@transfont@list}%
5988  \AddToHook{selectfont}{\bbl@transfont}% Hooks are global.
```

181

```
5989      \gdef\bbl@transfam{-unknown-}%
5990      \bbl@foreach\bbl@font@fams{%
5991        \AddToHook{##1family}{\def\bbl@transfam{##1}}%
5992        \bbl@ifsamestring{\@nameuse{##1default}}\familydefault
5993          {\xdef\bbl@transfam{##1}}%
5994          {}}}
5995  \DeclareRobustCommand\enablelocaletransform[1]{%
5996      \bbl@ifunset{bbl@ATR@#1@\languagename @}%
5997        {\bbl@error
5998          {'#1' for '\languagename' cannot be enabled.\\%
5999           Maybe there is a typo or it's a font-dependent transform}%
6000          {See the manual for further details.}}%
6001        {\bbl@csarg\setattribute{ATR@#1@\languagename @}\@ne}}
6002  \DeclareRobustCommand\disablelocaletransform[1]{%
6003      \bbl@ifunset{bbl@ATR@#1@\languagename @}%
6004        {\bbl@error
6005          {'#1' for '\languagename' cannot be disabled.\\%
6006           Maybe there is a typo or it's a font-dependent transform}%
6007          {See the manual for further details.}}%
6008        {\bbl@csarg\unsetattribute{ATR@#1@\languagename @}}}
6009  \def\bbl@activateposthyphen{%
6010    \let\bbl@activateposthyphen\relax
6011    \directlua{
6012      require('babel-transforms.lua')
6013      Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
6014    }}
6015  \def\bbl@activateprehyphen{%
6016    \let\bbl@activateprehyphen\relax
6017    \directlua{
6018      require('babel-transforms.lua')
6019      Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
6020    }}
```

12.10   Bidi

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

```
6021  \def\bbl@activate@preotf{%
6022    \let\bbl@activate@preotf\relax  % only once
6023    \directlua{
6024      Babel = Babel or {}
6025      %
6026      function Babel.pre_otfload_v(head)
6027        if Babel.numbers and Babel.digits_mapped then
6028          head = Babel.numbers(head)
6029        end
6030        if Babel.bidi_enabled then
6031          head = Babel.bidi(head, false, dir)
6032        end
6033        return head
6034      end
6035      %
6036      function Babel.pre_otfload_h(head, gc, sz, pt, dir)
6037        if Babel.numbers and Babel.digits_mapped then
6038          head = Babel.numbers(head)
6039        end
6040        if Babel.bidi_enabled then
6041          head = Babel.bidi(head, false, dir)
6042        end
6043        return head
6044      end
6045      %
```

```
6046     luatexbase.add_to_callback('pre_linebreak_filter',
6047       Babel.pre_otfload_v,
6048       'Babel.pre_otfload_v',
6049       luatexbase.priority_in_callback('pre_linebreak_filter',
6050         'luaotfload.node_processor') or nil)
6051     %
6052     luatexbase.add_to_callback('hpack_filter',
6053       Babel.pre_otfload_h,
6054       'Babel.pre_otfload_h',
6055       luatexbase.priority_in_callback('hpack_filter',
6056         'luaotfload.node_processor') or nil)
6057   }}
```

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=.

```
6058 \ifnum\bbl@bidimode>\@ne % Excludes default=1
6059   \let\bbl@beforeforeign\leavevmode
6060   \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
6061   \RequirePackage{luatexbase}
6062   \bbl@activate@preotf
6063   \directlua{
6064     require('babel-data-bidi.lua')
6065     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
6066       require('babel-bidi-basic.lua')
6067     \or
6068       require('babel-bidi-basic-r.lua')
6069     \fi}
6070   \newattribute\bbl@attr@dir
6071   \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
6072   \bbl@exp{\output{\bodydir\pagedir\the\output}}
6073 \fi
6074 \chardef\bbl@thetextdir\z@
6075 \chardef\bbl@thepardir\z@
6076 \def\bbl@getluadir#1{%
6077   \directlua{
6078     if tex.#1dir == 'TLT' then
6079       tex.sprint('0')
6080     elseif tex.#1dir == 'TRT' then
6081       tex.sprint('1')
6082     end}}
6083 \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
6084   \ifcase#3\relax
6085     \ifcase\bbl@getluadir{#1}\relax\else
6086       #2 TLT\relax
6087     \fi
6088   \else
6089     \ifcase\bbl@getluadir{#1}\relax
6090       #2 TRT\relax
6091     \fi
6092   \fi}
6093 % ..OOPPTT, with masks 0xC (par dir) and 0x3 (text dir)
6094 \def\bbl@thedir{0}
6095 \def\bbl@textdir#1{%
6096   \bbl@setluadir{text}\textdir{#1}%
6097   \chardef\bbl@thetextdir#1\relax
6098   \edef\bbl@thedir{\the\numexpr\bbl@thepardir*4+#1}%
6099   \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*4+#1}}
6100 \def\bbl@pardir#1{%  Used twice
6101   \bbl@setluadir{par}\pardir{#1}%
6102   \chardef\bbl@thepardir#1\relax}
6103 \def\bbl@bodydir{\bbl@setluadir{body}\bodydir}%   Used once
6104 \def\bbl@pagedir{\bbl@setluadir{page}\pagedir}%   Unused
```

```
6105 \def\bbl@dirparastext{\pardir\the\textdir\relax}% Used once
```

RTL text inside math needs special attention. It affects not only to actual math stuff, but also to 'tabular', which is based on a fake math.

```
6106 \ifnum\bbl@bidimode>\z@
6107   \def\bbl@insidemath{0}%
6108   \def\bbl@everymath{\def\bbl@insidemath{1}}
6109   \def\bbl@everydisplay{\def\bbl@insidemath{2}}
6110   \frozen@everymath\expandafter{%
6111     \expandafter\bbl@everymath\the\frozen@everymath}
6112   \frozen@everydisplay\expandafter{%
6113     \expandafter\bbl@everydisplay\the\frozen@everydisplay}
6114   \AtBeginDocument{
6115     \directlua{
6116       function Babel.math_box_dir(head)
6117         if not (token.get_macro('bbl@insidemath') == '0') then
6118           if Babel.hlist_has_bidi(head) then
6119             local d = node.new(node.id'dir')
6120             d.dir = '+TRT'
6121             node.insert_before(head, node.has_glyph(head), d)
6122             for item in node.traverse(head) do
6123               node.set_attribute(item,
6124                 Babel.attr_dir, token.get_macro('bbl@thedir'))
6125             end
6126           end
6127         end
6128         return head
6129       end
6130       luatexbase.add_to_callback("hpack_filter", Babel.math_box_dir,
6131         "Babel.math_box_dir", 0)
6132   }}%
6133 \fi
```

12.11   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.

```
6134 \bbl@trace{Redefinitions for bidi layout}
6135 %
6136 ⟨⟨*More package options⟩⟩ ≡
6137 \chardef\bbl@eqnpos\z@
6138 \DeclareOption{leqno}{\chardef\bbl@eqnpos\@ne}
6139 \DeclareOption{fleqn}{\chardef\bbl@eqnpos\tw@}
6140 ⟨⟨/More package options⟩⟩
6141 %
6142 \ifnum\bbl@bidimode>\z@
6143   \ifx\matheqdirmode\@undefined\else
6144     \matheqdirmode\@ne % A luatex primitive
6145   \fi
6146   \let\bbl@eqnodir\relax
6147   \def\bbl@eqdel{()}
6148   \def\bbl@eqnum{%
6149     {\normalfont\normalcolor
```

```
6150        \expandafter\@firstoftwo\bbl@eqdel
6151         \theequation
6152         \expandafter\@secondoftwo\bbl@eqdel}}
6153    \def\bbl@puteqno#1{\eqno\hbox{#1}}
6154    \def\bbl@putleqno#1{\leqno\hbox{#1}}
6155    \def\bbl@eqno@flip#1{%
6156      \ifdim\predisplaysize=-\maxdimen
6157        \eqno
6158        \hb@xt@.01pt{\hb@xt@\displaywidth{\hss{#1}}\hss}%
6159      \else
6160        \leqno\hbox{#1}%
6161      \fi}
6162    \def\bbl@leqno@flip#1{%
6163      \ifdim\predisplaysize=-\maxdimen
6164        \leqno
6165        \hb@xt@.01pt{\hss\hb@xt@\displaywidth{{#1}\hss}}%
6166      \else
6167        \eqno\hbox{#1}%
6168      \fi}
6169    \AtBeginDocument{%
6170      \ifx\bbl@noamsmath\relax\else
6171      \ifx\maketag@@@\@undefined % Normal equation, eqnarray
6172        \AddToHook{env/equation/begin}{%
6173          \ifnum\bbl@thetextdir>\z@
6174            \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6175            \let\@eqnnum\bbl@eqnum
6176            \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6177            \chardef\bbl@thetextdir\z@
6178            \bbl@add\normalfont{\bbl@eqnodir}%
6179            \ifcase\bbl@eqnpos
6180              \let\bbl@puteqno\bbl@eqno@flip
6181            \or
6182              \let\bbl@puteqno\bbl@leqno@flip
6183            \fi
6184          \fi}%
6185        \ifnum\bbl@eqnpos=\tw@\else
6186          \def\endequation{\bbl@puteqno{\@eqnnum}$$\@ignoretrue}%
6187        \fi
6188        \AddToHook{env/eqnarray/begin}{%
6189          \ifnum\bbl@thetextdir>\z@
6190            \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6191            \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6192            \chardef\bbl@thetextdir\z@
6193            \bbl@add\normalfont{\bbl@eqnodir}%
6194            \ifnum\bbl@eqnpos=\@ne
6195              \def\@eqnnum{%
6196                \setbox\z@\hbox{\bbl@eqnum}%
6197                \hbox to0.01pt{\hss\hbox to\displaywidth{\box\z@\hss}}}%
6198            \else
6199              \let\@eqnnum\bbl@eqnum
6200            \fi
6201          \fi}
6202        % Hack. YA luatex bug?:
6203        \expandafter\bbl@sreplace\csname] \endcsname{$$}{\eqno\kern.001pt$$}%
6204      \else % amstex
6205        \bbl@exp{% Hack to hide maybe undefined conditionals:
6206          \chardef\bbl@eqnpos=0%
6207            \<iftagsleft@>1\<else>\<if@fleqn>2\<fi>\<fi>\relax}%
6208        \ifnum\bbl@eqnpos=\@ne
6209          \let\bbl@ams@lap\hbox
6210        \else
6211          \let\bbl@ams@lap\llap
6212        \fi
```

```
6213      \ExplSyntaxOn
6214      \bbl@sreplace\intertext@{\normalbaselines}%
6215        {\normalbaselines
6216          \ifx\bbl@eqnodir\relax\else\bbl@pardir\@ne\bbl@eqnodir\fi}%
6217      \ExplSyntaxOff
6218      \def\bbl@ams@tagbox#1#2{#1{\bbl@eqnodir#2}}% #1=hbox|@lap|flip
6219      \ifx\bbl@ams@lap\hbox % leqno
6220        \def\bbl@ams@flip#1{%
6221          \hbox to 0.01pt{\hss\hbox to\displaywidth{{#1}\hss}}}%
6222      \else % eqno
6223        \def\bbl@ams@flip#1{%
6224          \hbox to 0.01pt{\hbox to\displaywidth{\hss{#1}}\hss}}%
6225      \fi
6226      \def\bbl@ams@preset#1{%
6227        \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6228        \ifnum\bbl@thetextdir>\z@
6229          \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6230          \bbl@sreplace\textdef@{\hbox}{\bbl@ams@tagbox\hbox}%
6231          \bbl@sreplace\maketag@@@{\hbox}{\bbl@ams@tagbox#1}%
6232        \fi}%
6233      \ifnum\bbl@eqnpos=\tw@\else
6234        \def\bbl@ams@equation{%
6235          \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6236          \ifnum\bbl@thetextdir>\z@
6237            \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6238            \chardef\bbl@thetextdir\z@
6239            \bbl@add\normalfont{\bbl@eqnodir}%
6240            \ifcase\bbl@eqnpos
6241              \def\veqno##1##2{\bbl@eqno@flip{##1##2}}%
6242            \or
6243              \def\veqno##1##2{\bbl@leqno@flip{##1##2}}%
6244            \fi
6245          \fi}%
6246        \AddToHook{env/equation/begin}{\bbl@ams@equation}%
6247        \AddToHook{env/equation*/begin}{\bbl@ams@equation}%
6248      \fi
6249      \AddToHook{env/cases/begin}{\bbl@ams@preset\bbl@ams@lap}%
6250      \AddToHook{env/multline/begin}{\bbl@ams@preset\hbox}%
6251      \AddToHook{env/gather/begin}{\bbl@ams@preset\bbl@ams@lap}%
6252      \AddToHook{env/gather*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6253      \AddToHook{env/align/begin}{\bbl@ams@preset\bbl@ams@lap}%
6254      \AddToHook{env/align*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6255      \AddToHook{env/eqnalign/begin}{\bbl@ams@preset\hbox}%
6256      % Hackish, for proper alignment. Don't ask me why it works!:
6257      \bbl@exp{% Avoid a 'visible' conditional
6258        \\\AddToHook{env/align*/end}{\<iftag@>\<else>\\\tag*{}\<fi>}}%
6259      \AddToHook{env/flalign/begin}{\bbl@ams@preset\hbox}%
6260      \AddToHook{env/split/before}{%
6261        \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6262        \ifnum\bbl@thetextdir>\z@
6263          \bbl@ifsamestring\@currenvir{equation}%
6264            {\ifx\bbl@ams@lap\hbox % leqno
6265              \def\bbl@ams@flip#1{%
6266                \hbox to 0.01pt{\hbox to\displaywidth{{#1}\hss}\hss}}%
6267            \else
6268              \def\bbl@ams@flip#1{%
6269                \hbox to 0.01pt{\hss\hbox to\displaywidth{\hss{#1}}}}%
6270            \fi}%
6271            {}%
6272        \fi}%
6273    \fi\fi}
6274 \fi
6275 \def\bbl@provide@extra#1{%
```

```
6276    % == Counters: mapdigits ==
6277    % Native digits
6278    \ifx\bbl@KVP@mapdigits\@nnil\else
6279      \bbl@ifunset{bbl@dgnat@\languagename}{}%
6280        {\RequirePackage{luatexbase}%
6281         \bbl@activate@preotf
6282         \directlua{
6283           Babel = Babel or {}  %%% -> presets in luababel
6284           Babel.digits_mapped = true
6285           Babel.digits = Babel.digits or {}
6286           Babel.digits[\the\localeid] =
6287             table.pack(string.utfvalue('\bbl@cl{dgnat}'))
6288           if not Babel.numbers then
6289             function Babel.numbers(head)
6290               local LOCALE = Babel.attr_locale
6291               local GLYPH = node.id'glyph'
6292               local inmath = false
6293               for item in node.traverse(head) do
6294                 if not inmath and item.id == GLYPH then
6295                   local temp = node.get_attribute(item, LOCALE)
6296                   if Babel.digits[temp] then
6297                     local chr = item.char
6298                     if chr > 47 and chr < 58 then
6299                       item.char = Babel.digits[temp][chr-47]
6300                     end
6301                   end
6302                 elseif item.id == node.id'math' then
6303                   inmath = (item.subtype == 0)
6304                 end
6305               end
6306               return head
6307             end
6308           end
6309         }}%
6310    \fi
6311    % == transforms ==
6312    \ifx\bbl@KVP@transforms\@nnil\else
6313      \def\bbl@elt##1##2##3{%
6314        \in@{$transforms.}{$##1}%
6315        \ifin@
6316          \def\bbl@tempa{##1}%
6317          \bbl@replace\bbl@tempa{transforms.}{}%
6318          \bbl@carg\bbl@transforms{babel\bbl@tempa}{##2}{##3}%
6319        \fi}%
6320      \csname bbl@inidata@\languagename\endcsname
6321      \bbl@release@transforms\relax % \relax closes the last item.
6322    \fi}
6323    % Start tabular here:
6324    \def\localerestoredirs{%
6325    \ifcase\bbl@thetextdir
6326      \ifnum\textdirection=\z@\else\textdir TLT\fi
6327    \else
6328      \ifnum\textdirection=\@ne\else\textdir TRT\fi
6329    \fi
6330    \ifcase\bbl@thepardir
6331      \ifnum\pardirection=\z@\else\pardir TLT\bodydir TLT\fi
6332    \else
6333      \ifnum\pardirection=\@ne\else\pardir TRT\bodydir TRT\fi
6334    \fi}
6335    \IfBabelLayout{tabular}%
6336    {\chardef\bbl@tabular@mode\tw@}% All RTL
6337    {\IfBabelLayout{notabular}%
6338      {\chardef\bbl@tabular@mode\z@}%
```

```
6339          {\chardef\bbl@tabular@mode\@ne}}% Mixed, with LTR cols
6340 \ifnum\bbl@bidimode>\@ne
6341   \ifnum\bbl@tabular@mode=\@ne
6342     \let\bbl@parabefore\relax
6343     \AddToHook{para/before}{\bbl@parabefore}
6344     \AtBeginDocument{%
6345       \bbl@replace\@tabular{$}{$%
6346         \def\bbl@insidemath{0}%
6347         \def\bbl@parabefore{\localerestoredirs}}%
6348       \ifnum\bbl@tabular@mode=\@ne
6349         \bbl@ifunset{@tabclassz}{}{%
6350           \bbl@exp{% Hide conditionals
6351             \\\bbl@sreplace\\\@tabclassz
6352               {\<ifcase>\\\@chnum}%
6353               {\\\localerestoredirs\<ifcase>\\\@chnum}}}%
6354         \@ifpackageloaded{colortbl}%
6355           {\bbl@sreplace\@classz
6356             {\hbox\bgroup\bgroup}{\hbox\bgroup\bgroup\localerestoredirs}}%
6357           {\@ifpackageloaded{array}%
6358             {\bbl@exp{% Hide conditionals
6359               \\\bbl@sreplace\\\@classz
6360                 {\<ifcase>\\\@chnum}%
6361                 {\bgroup\\\localerestoredirs\<ifcase>\\\@chnum}%
6362               \\\bbl@sreplace\\\@classz
6363                 {\\\do@row@strut\<fi>}{\\\do@row@strut\<fi>\egroup}}}%
6364             {}}%
6365     \fi}
6366   \fi
6367   \AtBeginDocument{%
6368     \@ifpackageloaded{multicol}%
6369       {\toks@\expandafter{\multi@column@out}%
6370        \edef\multi@column@out{\bodydir\pagedir\the\toks@}}%
6371       {}}
6372 \fi
6373 \ifx\bbl@opt@layout\@nnil\endinput\fi  % if no layout
```

OMEGA provided a companion to \mathdir (\nextfakemath) for those cases where we did not want it
to be applied, so that the writing direction of the main text was left unchanged. \bbl@nextfake is an
attempt to emulate it, because luatex has removed it without an alternative. Also, \hangindent does
not honour direction changes by default, so we need to redefine \@hangfrom.

```
6374 \ifnum\bbl@bidimode>\z@
6375   \def\bbl@nextfake#1{%  non-local changes, use always inside a group!
6376     \bbl@exp{%
6377       \def\\\bbl@insidemath{0}%
6378       \mathdir\the\bodydir
6379       #1%                 Once entered in math, set boxes to restore values
6380       \<ifmmode>%
6381         \everyvbox{%
6382           \the\everyvbox
6383           \bodydir\the\bodydir
6384           \mathdir\the\mathdir
6385           \everyhbox{\the\everyhbox}%
6386           \everyvbox{\the\everyvbox}}%
6387         \everyhbox{%
6388           \the\everyhbox
6389           \bodydir\the\bodydir
6390           \mathdir\the\mathdir
6391           \everyhbox{\the\everyhbox}%
6392           \everyvbox{\the\everyvbox}}%
6393       \<fi>}}%
6394   \def\@hangfrom#1{%
6395     \setbox\@tempboxa\hbox{{#1}}%
6396     \hangindent\wd\@tempboxa
```

```
6397        \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
6398          \shapemode\@ne
6399        \fi
6400        \noindent\box\@tempboxa}
6401 \fi
6402 \IfBabelLayout{tabular}
6403   {\let\bbl@OL@@tabular\@tabular
6404    \bbl@replace\@tabular{$}{\bbl@nextfake$}%
6405    \let\bbl@NL@@tabular\@tabular
6406    \AtBeginDocument{%
6407      \ifx\bbl@NL@@tabular\@tabular\else
6408        \bbl@replace\@tabular{$}{\bbl@nextfake$}%
6409        \let\bbl@NL@@tabular\@tabular
6410      \fi}}
6411    {}
6412 \IfBabelLayout{lists}
6413   {\let\bbl@OL@list\list
6414    \bbl@sreplace\list{\parshape}{\bbl@listparshape}%
6415    \let\bbl@NL@list\list
6416    \def\bbl@listparshape#1#2#3{%
6417      \parshape #1 #2 #3 %
6418      \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
6419        \shapemode\tw@
6420      \fi}}
6421    {}
6422 \IfBabelLayout{graphics}
6423   {\let\bbl@pictresetdir\relax
6424    \def\bbl@pictsetdir#1{%
6425      \ifcase\bbl@thetextdir
6426        \let\bbl@pictresetdir\relax
6427      \else
6428        \ifcase#1\bodydir TLT  % Remember this sets the inner boxes
6429          \or\textdir TLT
6430          \else\bodydir TLT \textdir TLT
6431        \fi
6432        % \(text|par)dir required in pgf:
6433        \def\bbl@pictresetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}%
6434      \fi}%
6435    \AddToHook{env/picture/begin}{\bbl@pictsetdir\tw@}%
6436    \directlua{
6437      Babel.get_picture_dir = true
6438      Babel.picture_has_bidi = 0
6439      %
6440      function Babel.picture_dir (head)
6441        if not Babel.get_picture_dir then return head end
6442        if Babel.hlist_has_bidi(head) then
6443          Babel.picture_has_bidi = 1
6444        end
6445        return head
6446      end
6447      luatexbase.add_to_callback("hpack_filter", Babel.picture_dir,
6448        "Babel.picture_dir")
6449    }%
6450    \AtBeginDocument{%
6451      \def\LS@rot{%
6452        \setbox\@outputbox\vbox{%
6453          \hbox dir TLT{\rotatebox{90}{\box\@outputbox}}}}%
6454      \long\def\put(#1,#2)#3{%
6455        \@killglue
6456        % Try:
6457        \ifx\bbl@pictresetdir\relax
6458          \def\bbl@tempc{0}%
6459        \else
```

189

```
6460          \directlua{
6461            Babel.get_picture_dir = true
6462            Babel.picture_has_bidi = 0
6463          }%
6464          \setbox\z@\hb@xt@\z@{%
6465            \@defaultunitsset\@tempdimc{#1}\unitlength
6466            \kern\@tempdimc
6467            #3\hss}% TODO: #3 executed twice (below). That's bad.
6468            \edef\bbl@tempc{\directlua{tex.print(Babel.picture_has_bidi)}}}%
6469          \fi
6470          % Do:
6471          \@defaultunitsset\@tempdimc{#2}\unitlength
6472          \raise\@tempdimc\hb@xt@\z@{%
6473            \@defaultunitsset\@tempdimc{#1}\unitlength
6474            \kern\@tempdimc
6475            {\ifnum\bbl@tempc>\z@\bbl@pictresetdir\fi#3}\hss}%
6476          \ignorespaces}%
6477      \MakeRobust\put}%
6478    \AtBeginDocument
6479      {\AddToHook{cmd/diagbox@pict/before}{\let\bbl@pictsetdir\@gobble}%
6480      \ifx\pgfpicture\@undefined\else % TODO. Allow deactivate?
6481        \AddToHook{env/pgfpicture/begin}{\bbl@pictsetdir\@ne}%
6482        \bbl@add\pgfinterruptpicture{\bbl@pictresetdir}%
6483        \bbl@add\pgfsys@beginpicture{\bbl@pictsetdir\z@}%
6484      \fi
6485      \ifx\tikzpicture\@undefined\else
6486        \AddToHook{env/tikzpicture/begin}{\bbl@pictsetdir\tw@}%
6487        \bbl@add\tikz@atbegin@node{\bbl@pictresetdir}%
6488        \bbl@sreplace\tikz{\begingroup}{\begingroup\bbl@pictsetdir\tw@}%
6489      \fi
6490      \ifx\tcolorbox\@undefined\else
6491        \def\tcb@drawing@env@begin{%
6492        \csname tcb@before@\tcb@split@state\endcsname
6493        \bbl@pictsetdir\tw@
6494        \begin{kvtcb@graphenv}%
6495        \tcb@bbdraw%
6496        \tcb@apply@graph@patches
6497        }%
6498        \def\tcb@drawing@env@end{%
6499        \end{kvtcb@graphenv}%
6500        \bbl@pictresetdir
6501        \csname tcb@after@\tcb@split@state\endcsname
6502        }%
6503      \fi
6504    }}
6505    {}
```

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.

```
6506 \IfBabelLayout{counters*}%
6507    {\bbl@add\bbl@opt@layout{.counters.}%
6508    \directlua
6509      luatexbase.add_to_callback("process_output_buffer",
6510        Babel.discard_sublr , "Babel.discard_sublr") }%
6511 }{}
6512 \IfBabelLayout{counters}%
6513    {\let\bbl@OL@@textsuperscript\@textsuperscript
6514    \bbl@sreplace\@textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
6515    \let\bbl@latinarabic=\@arabic
6516    \let\bbl@OL@@arabic\@arabic
6517    \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
6518    \@ifpackagewith{babel}{bidi=default}%
```

```
6519      {\let\bbl@asciiroman=\@roman
6520       \let\bbl@OL@@roman\@roman
6521       \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciiroman#1}}}%
6522       \let\bbl@asciiRoman=\@Roman
6523       \let\bbl@OL@@roman\@Roman
6524       \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}%
6525       \let\bbl@OL@labelenumii\labelenumii
6526       \def\labelenumii{)\theenumii(}%
6527       \let\bbl@OL@p@enumiii\p@enumiii
6528       \def\p@enumiii{\p@enumii)\theenumii(}}{}}{}
6529 ⟨⟨Footnote changes⟩⟩
6530 \IfBabelLayout{footnotes}%
6531   {\let\bbl@OL@footnote\footnote
6532    \BabelFootnote\footnote\languagename{}{}%
6533    \BabelFootnote\localfootnote\languagename{}{}%
6534    \BabelFootnote\mainfootnote{}{}{}}
6535   {}
```

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.

```
6536 \IfBabelLayout{extras}%
6537   {\let\bbl@OL@underline\underline
6538    \bbl@sreplace\underline{$\@@underline}{\bbl@nextfake$\@@underline}%
6539    \let\bbl@OL@LaTeX2e\LaTeX2e
6540    \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
6541      \if b\expandafter\@car\f@series\@nil\boldmath\fi
6542      \babelsublr{%
6543        \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}}}
6544   {}
6545 ⟨/luatex⟩
```

12.12   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.

```
6546 ⟨*transforms⟩
6547 Babel.linebreaking.replacements = {}
6548 Babel.linebreaking.replacements[0] = {}  -- pre
6549 Babel.linebreaking.replacements[1] = {}  -- post
6550 Babel.linebreaking.replacements[2] = {}  -- post-line WIP
6551
6552 -- Discretionaries contain strings as nodes
6553 function Babel.str_to_nodes(fn, matches, base)
6554   local n, head, last
6555   if fn == nil then return nil end
6556   for s in string.utfvalues(fn(matches)) do
6557     if base.id == 7 then
6558       base = base.replace
6559     end
6560     n = node.copy(base)
6561     n.char     = s
6562     if not head then
6563       head = n
6564     else
```

```
6565      last.next = n
6566    end
6567    last = n
6568  end
6569  return head
6570 end
6571
6572 Babel.fetch_subtext = {}
6573
6574 Babel.ignore_pre_char = function(node)
6575   return (node.lang == Babel.nohyphenation)
6576 end
6577
6578 -- Merging both functions doesn't seen feasible, because there are too
6579 -- many differences.
6580 Babel.fetch_subtext[0] = function(head)
6581   local word_string = ''
6582   local word_nodes = {}
6583   local lang
6584   local item = head
6585   local inmath = false
6586
6587   while item do
6588
6589     if item.id == 11 then
6590       inmath = (item.subtype == 0)
6591     end
6592
6593     if inmath then
6594       -- pass
6595
6596     elseif item.id == 29 then
6597       local locale = node.get_attribute(item, Babel.attr_locale)
6598
6599       if lang == locale or lang == nil then
6600         lang = lang or locale
6601         if Babel.ignore_pre_char(item) then
6602           word_string = word_string .. Babel.us_char
6603         else
6604           word_string = word_string .. unicode.utf8.char(item.char)
6605         end
6606         word_nodes[#word_nodes+1] = item
6607       else
6608         break
6609       end
6610
6611     elseif item.id == 12 and item.subtype == 13 then
6612       word_string = word_string .. ' '
6613       word_nodes[#word_nodes+1] = item
6614
6615     -- Ignore leading unrecognized nodes, too.
6616     elseif word_string ~= '' then
6617       word_string = word_string .. Babel.us_char
6618       word_nodes[#word_nodes+1] = item  -- Will be ignored
6619     end
6620
6621     item = item.next
6622   end
6623
6624   -- Here and above we remove some trailing chars but not the
6625   -- corresponding nodes. But they aren't accessed.
6626   if word_string:sub(-1) == ' ' then
6627     word_string = word_string:sub(1,-2)
```

```
6628      end
6629      word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
6630      return word_string, word_nodes, item, lang
6631    end
6632
6633    Babel.fetch_subtext[1] = function(head)
6634      local word_string = ''
6635      local word_nodes = {}
6636      local lang
6637      local item = head
6638      local inmath = false
6639
6640      while item do
6641
6642        if item.id == 11 then
6643          inmath = (item.subtype == 0)
6644        end
6645
6646        if inmath then
6647          -- pass
6648
6649        elseif item.id == 29 then
6650          if item.lang == lang or lang == nil then
6651            if (item.char ~= 124) and (item.char ~= 61) then -- not =, not |
6652              lang = lang or item.lang
6653              word_string = word_string .. unicode.utf8.char(item.char)
6654              word_nodes[#word_nodes+1] = item
6655            end
6656          else
6657            break
6658          end
6659
6660        elseif item.id == 7 and item.subtype == 2 then
6661          word_string = word_string .. '='
6662          word_nodes[#word_nodes+1] = item
6663
6664        elseif item.id == 7 and item.subtype == 3 then
6665          word_string = word_string .. '|'
6666          word_nodes[#word_nodes+1] = item
6667
6668        -- (1) Go to next word if nothing was found, and (2) implicitly
6669        -- remove leading USs.
6670        elseif word_string == '' then
6671          -- pass
6672
6673        -- This is the responsible for splitting by words.
6674        elseif (item.id == 12 and item.subtype == 13) then
6675          break
6676
6677        else
6678          word_string = word_string .. Babel.us_char
6679          word_nodes[#word_nodes+1] = item  -- Will be ignored
6680        end
6681
6682        item = item.next
6683      end
6684
6685      word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
6686      return word_string, word_nodes, item, lang
6687    end
6688
6689    function Babel.pre_hyphenate_replace(head)
6690      Babel.hyphenate_replace(head, 0)
```

```lua
6691 end
6692
6693 function Babel.post_hyphenate_replace(head)
6694   Babel.hyphenate_replace(head, 1)
6695 end
6696
6697 Babel.us_char = string.char(31)
6698
6699 function Babel.hyphenate_replace(head, mode)
6700   local u = unicode.utf8
6701   local lbkr = Babel.linebreaking.replacements[mode]
6702   if mode == 2 then mode = 0 end -- WIP
6703
6704   local word_head = head
6705
6706   while true do  -- for each subtext block
6707
6708     local w, w_nodes, nw, lang = Babel.fetch_subtext[mode](word_head)
6709
6710     if Babel.debug then
6711       print()
6712       print((mode == 0) and '@@@@<' or '@@@@>', w)
6713     end
6714
6715     if nw == nil and w == '' then break end
6716
6717     if not lang then goto next end
6718     if not lbkr[lang] then goto next end
6719
6720     -- For each saved (pre|post)hyphenation. TODO. Reconsider how
6721     -- loops are nested.
6722     for k=1, #lbkr[lang] do
6723       local p = lbkr[lang][k].pattern
6724       local r = lbkr[lang][k].replace
6725       local attr = lbkr[lang][k].attr or -1
6726
6727       if Babel.debug then
6728         print('*****', p, mode)
6729       end
6730
6731       -- This variable is set in some cases below to the first *byte*
6732       -- after the match, either as found by u.match (faster) or the
6733       -- computed position based on sc if w has changed.
6734       local last_match = 0
6735       local step = 0
6736
6737       -- For every match.
6738       while true do
6739         if Babel.debug then
6740           print('=====')
6741         end
6742         local new  -- used when inserting and removing nodes
6743
6744         local matches = { u.match(w, p, last_match) }
6745
6746         if #matches < 2 then break end
6747
6748         -- Get and remove empty captures (with ()'s, which return a
6749         -- number with the position), and keep actual captures
6750         -- (from (...)), if any, in matches.
6751         local first = table.remove(matches, 1)
6752         local last  = table.remove(matches, #matches)
6753         -- Non re-fetched substrings may contain \31, which separates
```

```lua
6754            -- subsubstrings.
6755            if string.find(w:sub(first, last-1), Babel.us_char) then break end
6756
6757            local save_last = last -- with A()BC()D, points to D
6758
6759            -- Fix offsets, from bytes to unicode. Explained above.
6760            first = u.len(w:sub(1, first-1)) + 1
6761            last  = u.len(w:sub(1, last-1)) -- now last points to C
6762
6763            -- This loop stores in a small table the nodes
6764            -- corresponding to the pattern. Used by 'data' to provide a
6765            -- predictable behavior with 'insert' (w_nodes is modified on
6766            -- the fly), and also access to 'remove'd nodes.
6767            local sc = first-1            -- Used below, too
6768            local data_nodes = {}
6769
6770            local enabled = true
6771            for q = 1, last-first+1 do
6772              data_nodes[q] = w_nodes[sc+q]
6773              if enabled
6774                  and attr > -1
6775                  and not node.has_attribute(data_nodes[q], attr)
6776                then
6777                  enabled = false
6778              end
6779            end
6780
6781            -- This loop traverses the matched substring and takes the
6782            -- corresponding action stored in the replacement list.
6783            -- sc = the position in substr nodes / string
6784            -- rc = the replacement table index
6785            local rc = 0
6786
6787            while rc < last-first+1 do -- for each replacement
6788              if Babel.debug then
6789                print('.....', rc + 1)
6790              end
6791              sc = sc + 1
6792              rc = rc + 1
6793
6794              if Babel.debug then
6795                Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6796                local ss = ''
6797                for itt in node.traverse(head) do
6798                 if itt.id == 29 then
6799                   ss = ss .. unicode.utf8.char(itt.char)
6800                 else
6801                   ss = ss .. '{' .. itt.id .. '}'
6802                 end
6803                end
6804                print('*****************', ss)
6805
6806              end
6807
6808              local crep = r[rc]
6809              local item = w_nodes[sc]
6810              local item_base = item
6811              local placeholder = Babel.us_char
6812              local d
6813
6814              if crep and crep.data then
6815                item_base = data_nodes[crep.data]
6816              end
```

195

```
6817
6818          if crep then
6819            step = crep.step or 0
6820          end
6821
6822          if (not enabled) or (crep and next(crep) == nil) then -- = {}
6823            last_match = save_last    -- Optimization
6824            goto next
6825
6826          elseif crep == nil or crep.remove then
6827            node.remove(head, item)
6828            table.remove(w_nodes, sc)
6829            w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
6830            sc = sc - 1  -- Nothing has been inserted.
6831            last_match = utf8.offset(w, sc+1+step)
6832            goto next
6833
6834          elseif crep and crep.kashida then -- Experimental
6835            node.set_attribute(item,
6836              Babel.attr_kashida,
6837              crep.kashida)
6838            last_match = utf8.offset(w, sc+1+step)
6839            goto next
6840
6841          elseif crep and crep.string then
6842            local str = crep.string(matches)
6843            if str == '' then  -- Gather with nil
6844              node.remove(head, item)
6845              table.remove(w_nodes, sc)
6846              w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
6847              sc = sc - 1  -- Nothing has been inserted.
6848            else
6849              local loop_first = true
6850              for s in string.utfvalues(str) do
6851                d = node.copy(item_base)
6852                d.char = s
6853                if loop_first then
6854                  loop_first = false
6855                  head, new = node.insert_before(head, item, d)
6856                  if sc == 1 then
6857                    word_head = head
6858                  end
6859                  w_nodes[sc] = d
6860                  w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc+1)
6861                else
6862                  sc = sc + 1
6863                  head, new = node.insert_before(head, item, d)
6864                  table.insert(w_nodes, sc, new)
6865                  w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc)
6866                end
6867                if Babel.debug then
6868                  print('.....', 'str')
6869                  Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6870                end
6871              end  -- for
6872              node.remove(head, item)
6873            end  -- if ''
6874            last_match = utf8.offset(w, sc+1+step)
6875            goto next
6876
6877          elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
6878            d = node.new(7, 3)   -- (disc, regular)
6879            d.pre     = Babel.str_to_nodes(crep.pre, matches, item_base)
```

```
6880            d.post    = Babel.str_to_nodes(crep.post, matches, item_base)
6881            d.replace = Babel.str_to_nodes(crep.no, matches, item_base)
6882            d.attr = item_base.attr
6883            if crep.pre == nil then  -- TeXbook p96
6884              d.penalty = crep.penalty or tex.hyphenpenalty
6885            else
6886              d.penalty = crep.penalty or tex.exhyphenpenalty
6887            end
6888            placeholder = '|'
6889            head, new = node.insert_before(head, item, d)
6890
6891          elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
6892            -- ERROR
6893
6894          elseif crep and crep.penalty then
6895            d = node.new(14, 0)   -- (penalty, userpenalty)
6896            d.attr = item_base.attr
6897            d.penalty = crep.penalty
6898            head, new = node.insert_before(head, item, d)
6899
6900          elseif crep and crep.space then
6901            -- 655360 = 10 pt = 10 * 65536 sp
6902            d = node.new(12, 13)     -- (glue, spaceskip)
6903            local quad = font.getfont(item_base.font).size or 655360
6904            node.setglue(d, crep.space[1] * quad,
6905                            crep.space[2] * quad,
6906                            crep.space[3] * quad)
6907            if mode == 0 then
6908              placeholder = ' '
6909            end
6910            head, new = node.insert_before(head, item, d)
6911
6912          elseif crep and crep.spacefactor then
6913            d = node.new(12, 13)     -- (glue, spaceskip)
6914            local base_font = font.getfont(item_base.font)
6915            node.setglue(d,
6916              crep.spacefactor[1] * base_font.parameters['space'],
6917              crep.spacefactor[2] * base_font.parameters['space_stretch'],
6918              crep.spacefactor[3] * base_font.parameters['space_shrink'])
6919            if mode == 0 then
6920              placeholder = ' '
6921            end
6922            head, new = node.insert_before(head, item, d)
6923
6924          elseif mode == 0 and crep and crep.space then
6925            -- ERROR
6926
6927          end  -- ie replacement cases
6928
6929          -- Shared by disc, space and penalty.
6930          if sc == 1 then
6931            word_head = head
6932          end
6933          if crep.insert then
6934            w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc)
6935            table.insert(w_nodes, sc, new)
6936            last = last + 1
6937          else
6938            w_nodes[sc] = d
6939            node.remove(head, item)
6940            w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1)
6941          end
6942
```

```
6943            last_match = utf8.offset(w, sc+1+step)

6944

6945            ::next::

6946

6947        end  -- for each replacement

6948

6949        if Babel.debug then

6950            print('.....', '/')

6951            Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)

6952        end

6953

6954      end  -- for match

6955

6956    end  -- for patterns

6957

6958    ::next::

6959    word_head = nw

6960  end  -- for substring

6961  return head

6962 end

6963

6964 -- This table stores capture maps, numbered consecutively

6965 Babel.capture_maps = {}

6966

6967 -- The following functions belong to the next macro

6968 function Babel.capture_func(key, cap)

6969  local ret = "[[" .. cap:gsub('{([0-9])}', "]]..m[%1]..[[") .. "]]"

6970  local cnt

6971  local u = unicode.utf8

6972  ret, cnt = ret:gsub('{([0-9])|([^|]+)|(.-)}', Babel.capture_func_map)

6973  if cnt == 0 then

6974    ret = u.gsub(ret, '{(%x%x%x%x+)}',

6975          function (n)

6976            return u.char(tonumber(n, 16))

6977          end)

6978  end

6979  ret = ret:gsub("%[%[%]%]%.%.", '')

6980  ret = ret:gsub("%.%.%[%[%]%]", '')

6981  return key .. [[=function(m) return ]] .. ret .. [[ end]]

6982 end

6983

6984 function Babel.capt_map(from, mapno)

6985  return Babel.capture_maps[mapno][from] or from

6986 end

6987

6988 -- Handle the {n|abc|ABC} syntax in captures

6989 function Babel.capture_func_map(capno, from, to)

6990  local u = unicode.utf8

6991  from = u.gsub(from, '{(%x%x%x%x+)}',

6992      function (n)

6993        return u.char(tonumber(n, 16))

6994      end)

6995  to = u.gsub(to, '{(%x%x%x%x+)}',

6996      function (n)

6997        return u.char(tonumber(n, 16))

6998      end)

6999  local froms = {}

7000  for s in string.utfcharacters(from) do

7001    table.insert(froms, s)

7002  end

7003  local cnt = 1

7004  table.insert(Babel.capture_maps, {})

7005  local mlen = table.getn(Babel.capture_maps)
```

```
7006   for s in string.utfcharacters(to) do
7007     Babel.capture_maps[mlen][froms[cnt]] = s
7008     cnt = cnt + 1
7009   end
7010   return "]]..Babel.capt_map(m[" .. capno .. "]," ..
7011         (mlen) .. ").." .. "[["
7012 end
7013
7014 -- Create/Extend reversed sorted list of kashida weights:
7015 function Babel.capture_kashida(key, wt)
7016   wt = tonumber(wt)
7017   if Babel.kashida_wts then
7018     for p, q in ipairs(Babel.kashida_wts) do
7019       if wt  == q then
7020         break
7021       elseif wt > q then
7022         table.insert(Babel.kashida_wts, p, wt)
7023         break
7024       elseif table.getn(Babel.kashida_wts) == p then
7025         table.insert(Babel.kashida_wts, wt)
7026       end
7027     end
7028   else
7029     Babel.kashida_wts = { wt }
7030   end
7031   return 'kashida = ' .. wt
7032 end
7033 ⟨/transforms⟩
```

12.13   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.

```
7034 ⟨∗basic-r⟩
7035 Babel = Babel or {}
7036
7037 Babel.bidi_enabled = true
7038
7039 require('babel-data-bidi.lua')
7040
7041 local characters = Babel.characters
7042 local ranges = Babel.ranges
7043
7044 local DIR = node.id("dir")
7045
7046 local function dir_mark(head, from, to, outer)
7047   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
7048   local d = node.new(DIR)
7049   d.dir = '+' .. dir
7050   node.insert_before(head, from, d)
7051   d = node.new(DIR)
7052   d.dir = '-' .. dir
7053   node.insert_after(head, to, d)
7054 end
7055
7056 function Babel.bidi(head, ispar)
7057   local first_n, last_n        -- first and last char with nums
7058   local last_es                -- an auxiliary 'last' used with nums
7059   local first_d, last_d        -- first and last char in L/R block
7060   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):

```
7061   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
7062   local strong_lr = (strong == 'l') and 'l' or 'r'
7063   local outer = strong
7064
7065   local new_dir = false
7066   local first_dir = false
7067   local inmath = false
7068
7069   local last_lr
7070
7071   local type_n = ''
7072
7073   for item in node.traverse(head) do
7074
7075     -- three cases: glyph, dir, otherwise
7076     if item.id == node.id'glyph'
7077       or (item.id == 7 and item.subtype == 2) then
7078
7079       local itemchar
7080       if item.id == 7 and item.subtype == 2 then
7081         itemchar = item.replace.char
7082       else
7083         itemchar = item.char
7084       end
7085       local chardata = characters[itemchar]
7086       dir = chardata and chardata.d or nil
7087       if not dir then
7088         for nn, et in ipairs(ranges) do
```

```
7089          if itemchar < et[1] then
7090            break
7091          elseif itemchar <= et[2] then
7092            dir = et[3]
7093            break
7094          end
7095        end
7096      end
7097      dir = dir or 'l'
7098      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).

```
7099      if new_dir then
7100        attr_dir = 0
7101        for at in node.traverse(item.attr) do
7102          if at.number == Babel.attr_dir then
7103            attr_dir = at.value & 0x3
7104          end
7105        end
7106        if attr_dir == 1 then
7107          strong = 'r'
7108        elseif attr_dir == 2 then
7109          strong = 'al'
7110        else
7111          strong = 'l'
7112        end
7113        strong_lr = (strong == 'l') and 'l' or 'r'
7114        outer = strong_lr
7115        new_dir = false
7116      end
7117
7118      if dir == 'nsm' then dir = strong end            -- W1
```

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

```
7119      dir_real = dir              -- We need dir_real to set strong below
7120      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:

```
7121      if strong == 'al' then
7122        if dir == 'en' then dir = 'an' end            -- W2
7123        if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
7124        strong_lr = 'r'                               -- W3
7125      end
```

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

```
7126    elseif item.id == node.id'dir' and not inmath then
7127      new_dir = true
7128      dir = nil
7129    elseif item.id == node.id'math' then
7130      inmath = (item.subtype == 0)
7131    else
7132      dir = nil          -- Not a char
7133    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>.

```
7134    if dir == 'en' or dir == 'an' or dir == 'et' then
7135      if dir ~= 'et' then
7136        type_n = dir
7137      end
7138      first_n = first_n or item
7139      last_n = last_es or item
7140      last_es = nil
7141    elseif dir == 'es' and last_n then -- W3+W6
7142      last_es = item
7143    elseif dir == 'cs' then            -- it's right - do nothing
7144    elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
7145      if strong_lr == 'r' and type_n ~= '' then
7146        dir_mark(head, first_n, last_n, 'r')
7147      elseif strong_lr == 'l' and first_d and type_n == 'an' then
7148        dir_mark(head, first_n, last_n, 'r')
7149        dir_mark(head, first_d, last_d, outer)
7150        first_d, last_d = nil, nil
7151      elseif strong_lr == 'l' and type_n ~= '' then
7152        last_d = last_n
7153      end
7154      type_n = ''
7155      first_n, last_n = nil, nil
7156    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:

```
7157    if dir == 'l' or dir == 'r' then
7158      if dir ~= outer then
7159        first_d = first_d or item
7160        last_d = item
7161      elseif first_d and dir ~= strong_lr then
7162        dir_mark(head, first_d, last_d, outer)
7163        first_d, last_d = nil, nil
7164      end
7165    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.

```
7166    if dir and not last_lr and dir ~= 'l' and outer == 'r' then
7167      item.char = characters[item.char] and
7168                  characters[item.char].m or item.char
7169    elseif (dir or new_dir) and last_lr ~= item then
7170      local mir = outer .. strong_lr .. (dir or outer)
7171      if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
7172        for ch in node.traverse(node.next(last_lr)) do
7173          if ch == item then break end
7174          if ch.id == node.id'glyph' and characters[ch.char] then
7175            ch.char = characters[ch.char].m or ch.char
7176          end
7177        end
7178      end
7179    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).

```
7180    if dir == 'l' or dir == 'r' then
7181      last_lr = item
7182      strong = dir_real              -- Don't search back - best save now
7183      strong_lr = (strong == 'l') and 'l' or 'r'
```

```
7184    elseif new_dir then
7185      last_lr = nil
7186    end
7187  end
```

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

```
7188  if last_lr and outer == 'r' then
7189    for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
7190      if characters[ch.char] then
7191        ch.char = characters[ch.char].m or ch.char
7192      end
7193    end
7194  end
7195  if first_n then
7196    dir_mark(head, first_n, last_n, outer)
7197  end
7198  if first_d then
7199    dir_mark(head, first_d, last_d, outer)
7200  end
```

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

```
7201  return node.prev(head) or head
7202 end
7203 ⟨/basic-r⟩
```

And here the Lua code for bidi=basic:

```
7204 ⟨∗basic⟩
7205 Babel = Babel or {}
7206
7207 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
7208
7209 Babel.fontmap = Babel.fontmap or {}
7210 Babel.fontmap[0] = {}       -- l
7211 Babel.fontmap[1] = {}       -- r
7212 Babel.fontmap[2] = {}       -- al/an
7213
7214 Babel.bidi_enabled = true
7215 Babel.mirroring_enabled = true
7216
7217 require('babel-data-bidi.lua')
7218
7219 local characters = Babel.characters
7220 local ranges = Babel.ranges
7221
7222 local DIR = node.id('dir')
7223 local GLYPH = node.id('glyph')
7224
7225 local function insert_implicit(head, state, outer)
7226  local new_state = state
7227  if state.sim and state.eim and state.sim ~= state.eim then
7228    dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
7229    local d = node.new(DIR)
7230    d.dir = '+' .. dir
7231    node.insert_before(head, state.sim, d)
7232    local d = node.new(DIR)
7233    d.dir = '-' .. dir
7234    node.insert_after(head, state.eim, d)
7235  end
7236  new_state.sim, new_state.eim = nil, nil
7237  return head, new_state
7238 end
7239
7240 local function insert_numeric(head, state)
```

```
7241    local new
7242    local new_state = state
7243    if state.san and state.ean and state.san ~= state.ean then
7244      local d = node.new(DIR)
7245      d.dir = '+TLT'
7246      _, new = node.insert_before(head, state.san, d)
7247      if state.san == state.sim then state.sim = new end
7248      local d = node.new(DIR)
7249      d.dir = '-TLT'
7250      _, new = node.insert_after(head, state.ean, d)
7251      if state.ean == state.eim then state.eim = new end
7252    end
7253    new_state.san, new_state.ean = nil, nil
7254    return head, new_state
7255 end
7256
7257 -- TODO - \hbox with an explicit dir can lead to wrong results
7258 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
7259 -- was s made to improve the situation, but the problem is the 3-dir
7260 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
7261 -- well.
7262
7263 function Babel.bidi(head, ispar, hdir)
7264    local d    -- d is used mainly for computations in a loop
7265    local prev_d = ''
7266    local new_d = false
7267
7268    local nodes = {}
7269    local outer_first = nil
7270    local inmath = false
7271
7272    local glue_d = nil
7273    local glue_i = nil
7274
7275    local has_en = false
7276    local first_et = nil
7277
7278    local has_hyperlink = false
7279
7280    local ATDIR = Babel.attr_dir
7281
7282    local save_outer
7283    local temp = node.get_attribute(head, ATDIR)
7284    if temp then
7285      temp = temp & 0x3
7286      save_outer = (temp == 0 and 'l') or
7287                   (temp == 1 and 'r') or
7288                   (temp == 2 and 'al')
7289    elseif ispar then              -- Or error? Shouldn't happen
7290      save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
7291    else                          -- Or error? Shouldn't happen
7292      save_outer = ('TRT' == hdir) and 'r' or 'l'
7293    end
7294      -- when the callback is called, we are just _after_ the box,
7295      -- and the textdir is that of the surrounding text
7296    -- if not ispar and hdir ~= tex.textdir then
7297    --   save_outer = ('TRT' == hdir) and 'r' or 'l'
7298    -- end
7299    local outer = save_outer
7300    local last = outer
7301    -- 'al' is only taken into account in the first, current loop
7302    if save_outer == 'al' then save_outer = 'r' end
7303
```

```
7304   local fontmap = Babel.fontmap
7305
7306   for item in node.traverse(head) do
7307
7308     -- In what follows, #node is the last (previous) node, because the
7309     -- current one is not added until we start processing the neutrals.
7310
7311     -- three cases: glyph, dir, otherwise
7312     if item.id == GLYPH
7313        or (item.id == 7 and item.subtype == 2) then
7314
7315       local d_font = nil
7316       local item_r
7317       if item.id == 7 and item.subtype == 2 then
7318         item_r = item.replace    -- automatic discs have just 1 glyph
7319       else
7320         item_r = item
7321       end
7322       local chardata = characters[item_r.char]
7323       d = chardata and chardata.d or nil
7324       if not d or d == 'nsm' then
7325         for nn, et in ipairs(ranges) do
7326           if item_r.char < et[1] then
7327             break
7328           elseif item_r.char <= et[2] then
7329             if not d then d = et[3]
7330             elseif d == 'nsm' then d_font = et[3]
7331             end
7332             break
7333           end
7334         end
7335       end
7336       d = d or 'l'
7337
7338       -- A short 'pause' in bidi for mapfont
7339       d_font = d_font or d
7340       d_font = (d_font == 'l' and 0) or
7341                (d_font == 'nsm' and 0) or
7342                (d_font == 'r' and 1) or
7343                (d_font == 'al' and 2) or
7344                (d_font == 'an' and 2) or nil
7345       if d_font and fontmap and fontmap[d_font][item_r.font] then
7346         item_r.font = fontmap[d_font][item_r.font]
7347       end
7348
7349       if new_d then
7350         table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
7351         if inmath then
7352           attr_d = 0
7353         else
7354           attr_d = node.get_attribute(item, ATDIR)
7355           attr_d = attr_d & 0x3
7356         end
7357         if attr_d == 1 then
7358           outer_first = 'r'
7359           last = 'r'
7360         elseif attr_d == 2 then
7361           outer_first = 'r'
7362           last = 'al'
7363         else
7364           outer_first = 'l'
7365           last = 'l'
7366         end
```

```lua
7367          outer = last
7368          has_en = false
7369          first_et = nil
7370          new_d = false
7371        end
7372
7373      if glue_d then
7374        if (d == 'l' and 'l' or 'r') ~= glue_d then
7375            table.insert(nodes, {glue_i, 'on', nil})
7376        end
7377        glue_d = nil
7378        glue_i = nil
7379      end
7380
7381    elseif item.id == DIR then
7382      d = nil
7383
7384      if head ~= item then new_d = true end
7385
7386    elseif item.id == node.id'glue' and item.subtype == 13 then
7387      glue_d = d
7388      glue_i = item
7389      d = nil
7390
7391    elseif item.id == node.id'math' then
7392      inmath = (item.subtype == 0)
7393
7394    elseif item.id == 8 and item.subtype == 19 then
7395      has_hyperlink = true
7396
7397    else
7398      d = nil
7399    end
7400
7401    -- AL <= EN/ET/ES      -- W2 + W3 + W6
7402    if last == 'al' and d == 'en' then
7403      d = 'an'              -- W3
7404    elseif last == 'al' and (d == 'et' or d == 'es') then
7405      d = 'on'             -- W6
7406    end
7407
7408    -- EN + CS/ES + EN      -- W4
7409    if d == 'en' and #nodes >= 2 then
7410      if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
7411          and nodes[#nodes-1][2] == 'en' then
7412        nodes[#nodes][2] = 'en'
7413      end
7414    end
7415
7416    -- AN + CS + AN          -- W4 too, because uax9 mixes both cases
7417    if d == 'an' and #nodes >= 2 then
7418      if (nodes[#nodes][2] == 'cs')
7419          and nodes[#nodes-1][2] == 'an' then
7420        nodes[#nodes][2] = 'an'
7421      end
7422    end
7423
7424    -- ET/EN                 -- W5 + W7->l / W6->on
7425    if d == 'et' then
7426      first_et = first_et or (#nodes + 1)
7427    elseif d == 'en' then
7428      has_en = true
7429      first_et = first_et or (#nodes + 1)
```

```
7430     elseif first_et then       -- d may be nil here !
7431       if has_en then
7432         if last == 'l' then
7433           temp = 'l'      -- W7
7434         else
7435           temp = 'en'    -- W5
7436         end
7437       else
7438         temp = 'on'      -- W6
7439       end
7440       for e = first_et, #nodes do
7441         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
7442       end
7443       first_et = nil
7444       has_en = false
7445     end
7446
7447     -- Force mathdir in math if ON (currently works as expected only
7448     -- with 'l')
7449     if inmath and d == 'on' then
7450       d = ('TRT' == tex.mathdir) and 'r' or 'l'
7451     end
7452
7453     if d then
7454       if d == 'al' then
7455         d = 'r'
7456         last = 'al'
7457       elseif d == 'l' or d == 'r' then
7458         last = d
7459       end
7460       prev_d = d
7461       table.insert(nodes, {item, d, outer_first})
7462     end
7463
7464     outer_first = nil
7465
7466   end
7467
7468   -- TODO -- repeated here in case EN/ET is the last node. Find a
7469   -- better way of doing things:
7470   if first_et then        -- dir may be nil here !
7471     if has_en then
7472       if last == 'l' then
7473         temp = 'l'      -- W7
7474       else
7475         temp = 'en'    -- W5
7476       end
7477     else
7478       temp = 'on'      -- W6
7479     end
7480     for e = first_et, #nodes do
7481       if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
7482     end
7483   end
7484
7485   -- dummy node, to close things
7486   table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
7487
7488   --------------  NEUTRAL ----------------
7489
7490   outer = save_outer
7491   last = outer
7492
```

```
7493  local first_on = nil
7494
7495  for q = 1, #nodes do
7496    local item
7497
7498    local outer_first = nodes[q][3]
7499    outer = outer_first or outer
7500    last = outer_first or last
7501
7502    local d = nodes[q][2]
7503    if d == 'an' or d == 'en' then d = 'r' end
7504    if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
7505
7506    if d == 'on' then
7507      first_on = first_on or q
7508    elseif first_on then
7509      if last == d then
7510        temp = d
7511      else
7512        temp = outer
7513      end
7514      for r = first_on, q - 1 do
7515        nodes[r][2] = temp
7516        item = nodes[r][1]     -- MIRRORING
7517        if Babel.mirroring_enabled and item.id == GLYPH
7518            and temp == 'r' and characters[item.char] then
7519          local font_mode = ''
7520          if item.font > 0 and font.fonts[item.font].properties then
7521            font_mode = font.fonts[item.font].properties.mode
7522          end
7523          if font_mode ~= 'harf' and font_mode ~= 'plug' then
7524            item.char = characters[item.char].m or item.char
7525          end
7526        end
7527      end
7528      first_on = nil
7529    end
7530
7531    if d == 'r' or d == 'l' then last = d end
7532  end
7533
7534  -------------  IMPLICIT, REORDER ----------------
7535
7536  outer = save_outer
7537  last = outer
7538
7539  local state = {}
7540  state.has_r = false
7541
7542  for q = 1, #nodes do
7543
7544    local item = nodes[q][1]
7545
7546    outer = nodes[q][3] or outer
7547
7548    local d = nodes[q][2]
7549
7550    if d == 'nsm' then d = last end                -- W1
7551    if d == 'en' then d = 'an' end
7552    local isdir = (d == 'r' or d == 'l')
7553
7554    if outer == 'l' and d == 'an' then
7555      state.san = state.san or item
```

```
7556        state.ean = item
7557      elseif state.san then
7558        head, state = insert_numeric(head, state)
7559      end
7560
7561      if outer == 'l' then
7562        if d == 'an' or d == 'r' then      -- im -> implicit
7563          if d == 'r' then state.has_r = true end
7564          state.sim = state.sim or item
7565          state.eim = item
7566        elseif d == 'l' and state.sim and state.has_r then
7567          head, state = insert_implicit(head, state, outer)
7568        elseif d == 'l' then
7569          state.sim, state.eim, state.has_r = nil, nil, false
7570        end
7571      else
7572        if d == 'an' or d == 'l' then
7573          if nodes[q][3] then -- nil except after an explicit dir
7574            state.sim = item  -- so we move sim 'inside' the group
7575          else
7576            state.sim = state.sim or item
7577          end
7578          state.eim = item
7579        elseif d == 'r' and state.sim then
7580          head, state = insert_implicit(head, state, outer)
7581        elseif d == 'r' then
7582          state.sim, state.eim = nil, nil
7583        end
7584      end
7585
7586      if isdir then
7587        last = d            -- Don't search back - best save now
7588      elseif d == 'on' and state.san  then
7589        state.san = state.san or item
7590        state.ean = item
7591      end
7592
7593    end
7594
7595    head = node.prev(head) or head
7596
7597    ------------- FIX HYPERLINKS ----------------
7598
7599    if has_hyperlink then
7600      local flag, linking = 0, 0
7601      for item in node.traverse(head) do
7602        if item.id == DIR then
7603          if item.dir == '+TRT' or item.dir == '+TLT' then
7604            flag = flag + 1
7605          elseif item.dir == '-TRT' or item.dir == '-TLT' then
7606            flag = flag - 1
7607          end
7608        elseif item.id == 8 and item.subtype == 19 then
7609          linking = flag
7610        elseif item.id == 8 and item.subtype == 20 then
7611          if linking > 0 then
7612            if item.prev.id == DIR and
7613                (item.prev.dir == '-TRT' or item.prev.dir == '-TLT') then
7614              d = node.new(DIR)
7615              d.dir = item.prev.dir
7616              node.remove(head, item.prev)
7617              node.insert_after(head, item, d)
7618            end
```

```
7619          end
7620        linking = 0
7621      end
7622    end
7623  end
7624
7625  return head
7626 end
```
7627 ⟨/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.

7628 ⟨∗nil⟩
7629 \ProvidesLanguage{nil}[⟨⟨date⟩⟩ ⟨⟨version⟩⟩ Nil language]
7630 \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.

7631 \ifx\l@nil\@undefined
7632   \newlanguage\l@nil
7633   \@namedef{bbl@hyphendata@\the\l@nil}{{}{}}% Remove warning
7634   \let\bbl@elt\relax
7635   \edef\bbl@languages{% Add it to the list of languages
7636   \bbl@languages\bbl@elt{nil}{\the\l@nil}{}{}}
7637 \fi

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

7638 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}

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

\captionnil
\datenil 7639 \let\captionsnil\@empty
7640 \let\datenil\@empty

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

7641 \def\bbl@inidata@nil{%
7642   \bbl@elt{identification}{tag.ini}{und}%
7643   \bbl@elt{identification}{load.level}{0}%
7644   \bbl@elt{identification}{charset}{utf8}%
7645   \bbl@elt{identification}{version}{1.0}%
7646   \bbl@elt{identification}{date}{2022-05-16}%
7647   \bbl@elt{identification}{name.local}{nil}%

```
7648    \bbl@elt{identification}{name.english}{nil}%
7649    \bbl@elt{identification}{name.babel}{nil}%
7650    \bbl@elt{identification}{tag.bcp47}{und}%
7651    \bbl@elt{identification}{language.tag.bcp47}{und}%
7652    \bbl@elt{identification}{tag.opentype}{dflt}%
7653    \bbl@elt{identification}{script.name}{Latin}%
7654    \bbl@elt{identification}{script.tag.bcp47}{Latn}%
7655    \bbl@elt{identification}{script.tag.opentype}{DFLT}%
7656    \bbl@elt{identification}{level}{1}%
7657    \bbl@elt{identification}{encodings}{}%
7658    \bbl@elt{identification}{derivate}{no}}
7659 \@namedef{bbl@tbcp@nil}{und}
7660 \@namedef{bbl@lbcp@nil}{und}
7661 \@namedef{bbl@lotf@nil}{dflt}
7662 \@namedef{bbl@elname@nil}{nil}
7663 \@namedef{bbl@lname@nil}{nil}
7664 \@namedef{bbl@esname@nil}{Latin}
7665 \@namedef{bbl@sname@nil}{Latin}
7666 \@namedef{bbl@sbcp@nil}{Latn}
7667 \@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.

```
7668 \ldf@finish{nil}
7669 ⟨/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.

```
7670 ⟨⟨*Compute Julian day⟩⟩ ≡
7671 \def\bbl@fpmod#1#2{(#1-#2*floor(#1/#2))}
7672 \def\bbl@cs@gregleap#1{%
7673    (\bbl@fpmod{#1}{4} == 0) &&
7674      (!((\bbl@fpmod{#1}{100} == 0) && (\bbl@fpmod{#1}{400} != 0)))}
7675 \def\bbl@cs@jd#1#2#3{% year, month, day
7676   \fp_eval:n{ 1721424.5   + (365 * (#1 - 1)) +
7677     floor((#1 - 1) / 4)   + (-floor((#1 - 1) / 100)) +
7678     floor((#1 - 1) / 400) + floor((((367 * #2) - 362) / 12) +
7679     ((#2 <= 2) ? 0 : (\bbl@cs@gregleap{#1} ? -1 : -2)) + #3) }}
7680 ⟨⟨/Compute Julian day⟩⟩
```

15.1   Islamic

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

```
7681 ⟨*ca-islamic⟩
7682 \ExplSyntaxOn
7683 ⟨⟨Compute Julian day⟩⟩
7684 % == islamic (default)
7685 % Not yet implemented
7686 \def\bbl@ca@islamic#1-#2-#3\@@#4#5#6{}
```

The Civil calendar.

```
7687 \def\bbl@cs@isltojd#1#2#3{ % year, month, day
7688    ((#3 + ceil(29.5 * (#2 - 1)) +
7689    (#1 - 1) * 354 + floor((3 + (11 * #1)) / 30) +
7690    1948439.5) - 1) }
7691 \@namedef{bbl@ca@islamic-civil++}{\bbl@ca@islamicvl@x{+2}}
7692 \@namedef{bbl@ca@islamic-civil+}{\bbl@ca@islamicvl@x{+1}}
7693 \@namedef{bbl@ca@islamic-civil}{\bbl@ca@islamicvl@x{}}
```

```
7694 \@namedef{bbl@ca@islamic-civil-}{\bbl@ca@islamicvl@x{-1}}
7695 \@namedef{bbl@ca@islamic-civil--}{\bbl@ca@islamicvl@x{-2}}
7696 \def\bbl@ca@islamicvl@x#1#2-#3-#4\@@#5#6#7{%
7697   \edef\bbl@tempa{%
7698     \fp_eval:n{ floor(\bbl@cs@jd{#2}{#3}{#4})+0.5 #1}}%
7699   \edef#5{%
7700     \fp_eval:n{ floor(((30*(\bbl@tempa-1948439.5)) + 10646)/10631) }}%
7701   \edef#6{\fp_eval:n{
7702     min(12,ceil((\bbl@tempa-(29+\bbl@cs@isltojd{#5}{1}{1}))/29.5)+1) }}%
7703   \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).

```
7704 \def\bbl@cs@umalqura@data{56660, 56690,56719,56749,56778,56808,%
7705   56837,56867,56897,56926,56956,56985,57015,57044,57074,57103,%
7706   57133,57162,57192,57221,57251,57280,57310,57340,57369,57399,%
7707   57429,57458,57487,57517,57546,57576,57605,57634,57664,57694,%
7708   57723,57753,57783,57813,57842,57871,57901,57930,57959,57989,%
7709   58018,58048,58077,58107,58137,58167,58196,58226,58255,58285,%
7710   58314,58343,58373,58402,58432,58461,58491,58521,58551,58580,%
7711   58610,58639,58669,58698,58727,58757,58786,58816,58845,58875,%
7712   58905,58934,58964,58994,59023,59053,59082,59111,59141,59170,%
7713   59200,59229,59259,59288,59318,59348,59377,59407,59436,59466,%
7714   59495,59525,59554,59584,59613,59643,59672,59702,59731,59761,%
7715   59791,59820,59850,59879,59909,59939,59968,59997,60027,60056,%
7716   60086,60115,60145,60174,60204,60234,60264,60293,60323,60352,%
7717   60381,60411,60440,60469,60499,60528,60558,60588,60618,60648,%
7718   60677,60707,60736,60765,60795,60824,60853,60883,60912,60942,%
7719   60972,61002,61031,61061,61090,61120,61149,61179,61208,61237,%
7720   61267,61296,61326,61356,61385,61415,61445,61474,61504,61533,%
7721   61563,61592,61621,61651,61680,61710,61739,61769,61799,61828,%
7722   61858,61888,61917,61947,61976,62006,62035,62064,62094,62123,%
7723   62153,62182,62212,62242,62271,62301,62331,62360,62390,62419,%
7724   62448,62478,62507,62537,62566,62596,62625,62655,62685,62715,%
7725   62744,62774,62803,62832,62862,62891,62921,62950,62980,63009,%
7726   63039,63069,63099,63128,63157,63187,63216,63246,63275,63305,%
7727   63334,63363,63393,63423,63453,63482,63512,63541,63571,63600,%
7728   63630,63659,63689,63718,63747,63777,63807,63836,63866,63895,%
7729   63925,63955,63984,64014,64043,64073,64102,64131,64161,64190,%
7730   64220,64249,64279,64309,64339,64368,64398,64427,64457,64486,%
7731   64515,64545,64574,64603,64633,64663,64692,64722,64752,64782,%
7732   64811,64841,64870,64899,64929,64958,64987,65017,65047,65076,%
7733   65106,65136,65166,65195,65225,65254,65283,65313,65342,65371,%
7734   65401,65431,65460,65490,65520}
7735 \@namedef{bbl@ca@islamic-umalqura+}{\bbl@ca@islamcuqr@x{+1}}
7736 \@namedef{bbl@ca@islamic-umalqura}{\bbl@ca@islamcuqr@x{}}
7737 \@namedef{bbl@ca@islamic-umalqura-}{\bbl@ca@islamcuqr@x{-1}}
7738 \def\bbl@ca@islamcuqr@x#1#2-#3-#4\@@#5#6#7{%
7739   \ifnum#2>2014 \ifnum#2<2038
7740     \bbl@afterfi\expandafter\@gobble
7741   \fi\fi
7742     {\bbl@error{Year~out~of~range}{The~allowed~range~is~2014-2038}}%
7743   \edef\bbl@tempd{\fp_eval:n{ % (Julian) day
7744     \bbl@cs@jd{#2}{#3}{#4} + 0.5 - 2400000 #1}}%
7745   \count@\@ne
7746   \bbl@foreach\bbl@cs@umalqura@data{%
7747     \advance\count@\@ne
7748     \ifnum##1>\bbl@tempd\else
7749       \edef\bbl@tempe{\the\count@}%
7750       \edef\bbl@tempb{##1}%
7751     \fi}%
```

```
7752    \edef\bbl@templ{\fp_eval:n{ \bbl@tempe + 16260 + 949 }}% month~lunar
7753    \edef\bbl@tempa{\fp_eval:n{ floor((\bbl@templ - 1 ) / 12) }}% annus
7754    \edef#5{\fp_eval:n{ \bbl@tempa + 1  }}%
7755    \edef#6{\fp_eval:n{ \bbl@templ - (12 * \bbl@tempa) }}%
7756    \edef#7{\fp_eval:n{ \bbl@tempd - \bbl@tempb + 1 }}}}
7757 \ExplSyntaxOff
7758 \bbl@add\bbl@precalendar{%
7759    \bbl@replace\bbl@ld@calendar{-civil}{}%
7760    \bbl@replace\bbl@ld@calendar{-umalqura}{}%
7761    \bbl@replace\bbl@ld@calendar{+}{}%
7762    \bbl@replace\bbl@ld@calendar{-}{}}
7763 ⟨/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`

```
7764 ⟨*ca-hebrew⟩
7765 \newcount\bbl@cntcommon
7766 \def\bbl@remainder#1#2#3{%
7767    #3=#1\relax
7768    \divide #3 by #2\relax
7769    \multiply #3 by -#2\relax
7770    \advance #3 by #1\relax}%
7771 \newif\ifbbl@divisible
7772 \def\bbl@checkifdivisible#1#2{%
7773    {\countdef\tmp=0
7774     \bbl@remainder{#1}{#2}{\tmp}%
7775     \ifnum \tmp=0
7776         \global\bbl@divisibletrue
7777     \else
7778         \global\bbl@divisiblefalse
7779     \fi}}
7780 \newif\ifbbl@gregleap
7781 \def\bbl@ifgregleap#1{%
7782    \bbl@checkifdivisible{#1}{4}%
7783    \ifbbl@divisible
7784        \bbl@checkifdivisible{#1}{100}%
7785        \ifbbl@divisible
7786            \bbl@checkifdivisible{#1}{400}%
7787            \ifbbl@divisible
7788                \bbl@gregleaptrue
7789            \else
7790                \bbl@gregleapfalse
7791            \fi
7792        \else
7793            \bbl@gregleaptrue
7794        \fi
7795    \else
7796        \bbl@gregleapfalse
7797    \fi
7798    \ifbbl@gregleap}
7799 \def\bbl@gregdayspriormonths#1#2#3{%
7800    {#3=\ifcase #1 0 \or 0 \or 31 \or 59 \or 90 \or 120 \or 151 \or
7801        181 \or 212 \or 243 \or 273 \or 304 \or 334 \fi
7802    \bbl@ifgregleap{#2}%
7803        \ifnum #1 > 2
7804            \advance #3 by 1
7805        \fi
7806    \fi
7807    \global\bbl@cntcommon=#3}%
```

```
7808      #3=\bbl@cntcommon}
7809 \def\bbl@gregdaysprioryears#1#2{%
7810   {\countdef\tmpc=4
7811    \countdef\tmpb=2
7812    \tmpb=#1\relax
7813    \advance \tmpb by -1
7814    \tmpc=\tmpb
7815    \multiply \tmpc by 365
7816    #2=\tmpc
7817    \tmpc=\tmpb
7818    \divide \tmpc by 4
7819    \advance #2 by \tmpc
7820    \tmpc=\tmpb
7821    \divide \tmpc by 100
7822    \advance #2 by -\tmpc
7823    \tmpc=\tmpb
7824    \divide \tmpc by 400
7825    \advance #2 by \tmpc
7826    \global\bbl@cntcommon=#2\relax}%
7827   #2=\bbl@cntcommon}
7828 \def\bbl@absfromgreg#1#2#3#4{%
7829   {\countdef\tmpd=0
7830    #4=#1\relax
7831    \bbl@gregdayspriormonths{#2}{#3}{\tmpd}%
7832    \advance #4 by \tmpd
7833    \bbl@gregdaysprioryears{#3}{\tmpd}%
7834    \advance #4 by \tmpd
7835    \global\bbl@cntcommon=#4\relax}%
7836   #4=\bbl@cntcommon}
7837 \newif\ifbbl@hebrleap
7838 \def\bbl@checkleaphebryear#1{%
7839   {\countdef\tmpa=0
7840    \countdef\tmpb=1
7841    \tmpa=#1\relax
7842    \multiply \tmpa by 7
7843    \advance \tmpa by 1
7844    \bbl@remainder{\tmpa}{19}{\tmpb}%
7845    \ifnum \tmpb < 7
7846        \global\bbl@hebrleaptrue
7847    \else
7848        \global\bbl@hebrleapfalse
7849    \fi}}
7850 \def\bbl@hebrelapsedmonths#1#2{%
7851   {\countdef\tmpa=0
7852    \countdef\tmpb=1
7853    \countdef\tmpc=2
7854    \tmpa=#1\relax
7855    \advance \tmpa by -1
7856    #2=\tmpa
7857    \divide #2 by 19
7858    \multiply #2 by 235
7859    \bbl@remainder{\tmpa}{19}{\tmpb}% \tmpa=years%19-years this cycle
7860    \tmpc=\tmpb
7861    \multiply \tmpb by 12
7862    \advance #2 by \tmpb
7863    \multiply \tmpc by 7
7864    \advance \tmpc by 1
7865    \divide \tmpc by 19
7866    \advance #2 by \tmpc
7867    \global\bbl@cntcommon=#2}%
7868   #2=\bbl@cntcommon}
7869 \def\bbl@hebrelapseddays#1#2{%
7870   {\countdef\tmpa=0
```

```
7871    \countdef\tmpb=1
7872    \countdef\tmpc=2
7873    \bbl@hebrelapsedmonths{#1}{#2}%
7874    \tmpa=#2\relax
7875    \multiply \tmpa by 13753
7876    \advance \tmpa by 5604
7877    \bbl@remainder{\tmpa}{25920}{\tmpc}% \tmpc == ConjunctionParts
7878    \divide \tmpa by 25920
7879    \multiply #2 by 29
7880    \advance #2 by 1
7881    \advance #2 by \tmpa
7882    \bbl@remainder{#2}{7}{\tmpa}%
7883    \ifnum \tmpc < 19440
7884        \ifnum \tmpc < 9924
7885        \else
7886            \ifnum \tmpa=2
7887                \bbl@checkleaphebryear{#1}% of a common year
7888                \ifbbl@hebrleap
7889                \else
7890                    \advance #2 by 1
7891                \fi
7892            \fi
7893        \fi
7894        \ifnum \tmpc < 16789
7895        \else
7896            \ifnum \tmpa=1
7897                \advance #1 by -1
7898                \bbl@checkleaphebryear{#1}% at the end of leap year
7899                \ifbbl@hebrleap
7900                    \advance #2 by 1
7901                \fi
7902            \fi
7903        \fi
7904    \else
7905        \advance #2 by 1
7906    \fi
7907    \bbl@remainder{#2}{7}{\tmpa}%
7908    \ifnum \tmpa=0
7909        \advance #2 by 1
7910    \else
7911        \ifnum \tmpa=3
7912            \advance #2 by 1
7913        \else
7914            \ifnum \tmpa=5
7915                \advance #2 by 1
7916            \fi
7917        \fi
7918    \fi
7919    \global\bbl@cntcommon=#2\relax}%
7920    #2=\bbl@cntcommon}
7921 \def\bbl@daysinhebryear#1#2{%
7922    {\countdef\tmpe=12
7923    \bbl@hebrelapseddays{#1}{\tmpe}%
7924    \advance #1 by 1
7925    \bbl@hebrelapseddays{#1}{#2}%
7926    \advance #2 by -\tmpe
7927    \global\bbl@cntcommon=#2}%
7928    #2=\bbl@cntcommon}
7929 \def\bbl@hebrdayspriormonths#1#2#3{%
7930    {\countdef\tmpf= 14
7931    #3=\ifcase #1\relax
7932            0 \or
7933            0 \or
```

```
7934          30 \or
7935          59 \or
7936          89 \or
7937         118 \or
7938         148 \or
7939         148 \or
7940         177 \or
7941         207 \or
7942         236 \or
7943         266 \or
7944         295 \or
7945         325 \or
7946         400
7947     \fi
7948     \bbl@checkleaphebryear{#2}%
7949     \ifbbl@hebrleap
7950         \ifnum #1 > 6
7951             \advance #3 by 30
7952         \fi
7953     \fi
7954     \bbl@daysinhebryear{#2}{\tmpf}%
7955     \ifnum #1 > 3
7956         \ifnum \tmpf=353
7957             \advance #3 by -1
7958         \fi
7959         \ifnum \tmpf=383
7960             \advance #3 by -1
7961         \fi
7962     \fi
7963     \ifnum #1 > 2
7964         \ifnum \tmpf=355
7965             \advance #3 by 1
7966         \fi
7967         \ifnum \tmpf=385
7968             \advance #3 by 1
7969         \fi
7970     \fi
7971     \global\bbl@cntcommon=#3\relax}%
7972   #3=\bbl@cntcommon}
7973 \def\bbl@absfromhebr#1#2#3#4{%
7974   {#4=#1\relax
7975   \bbl@hebrdayspriormonths{#2}{#3}{#1}%
7976   \advance #4 by #1\relax
7977   \bbl@hebrelapseddays{#3}{#1}%
7978   \advance #4 by #1\relax
7979   \advance #4 by -1373429
7980   \global\bbl@cntcommon=#4\relax}%
7981   #4=\bbl@cntcommon}
7982 \def\bbl@hebrfromgreg#1#2#3#4#5#6{%
7983   {\countdef\tmpx= 17
7984   \countdef\tmpy= 18
7985   \countdef\tmpz= 19
7986   #6=#3\relax
7987   \global\advance #6 by 3761
7988   \bbl@absfromgreg{#1}{#2}{#3}{#4}%
7989   \tmpz=1  \tmpy=1
7990   \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
7991   \ifnum \tmpx > #4\relax
7992       \global\advance #6 by -1
7993       \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
7994   \fi
7995   \advance #4 by -\tmpx
7996   \advance #4 by 1
```

```
7997    #5=#4\relax
7998    \divide #5 by 30
7999    \loop
8000       \bbl@hebrdayspriormonths{#5}{#6}{\tmpx}%
8001       \ifnum \tmpx < #4\relax
8002          \advance #5 by 1
8003          \tmpy=\tmpx
8004    \repeat
8005    \global\advance #5 by -1
8006    \global\advance #4 by -\tmpy}}
8007 \newcount\bbl@hebrday \newcount\bbl@hebrmonth \newcount\bbl@hebryear
8008 \newcount\bbl@gregday \newcount\bbl@gregmonth \newcount\bbl@gregyear
8009 \def\bbl@ca@hebrew#1-#2-#3\@@#4#5#6{%
8010    \bbl@gregday=#3\relax \bbl@gregmonth=#2\relax \bbl@gregyear=#1\relax
8011    \bbl@hebrfromgreg
8012       {\bbl@gregday}{\bbl@gregmonth}{\bbl@gregyear}%
8013       {\bbl@hebrday}{\bbl@hebrmonth}{\bbl@hebryear}%
8014    \edef#4{\the\bbl@hebryear}%
8015    \edef#5{\the\bbl@hebrmonth}%
8016    \edef#6{\the\bbl@hebrday}}
8017 ⟨/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).

```
8018 ⟨*ca-persian⟩
8019 \ExplSyntaxOn
8020 ⟨⟨Compute Julian day⟩⟩
8021 \def\bbl@cs@firstjal@xx{2012,2016,2020,2024,2028,2029,% March 20
8022    2032,2033,2036,2037,2040,2041,2044,2045,2048,2049}
8023 \def\bbl@ca@persian#1-#2-#3\@@#4#5#6{%
8024    \edef\bbl@tempa{#1}%  20XX-03-\bbl@tempe = 1 farvardin:
8025    \ifnum\bbl@tempa>2012 \ifnum\bbl@tempa<2051
8026       \bbl@afterfi\expandafter\@gobble
8027    \fi\fi
8028       {\bbl@error{Year~out~of~range}{The~allowed~range~is~2013-2050}}%
8029    \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
8030    \ifin@\def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
8031    \edef\bbl@tempc{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{#2}{#3}+.5}}% current
8032    \edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}% begin
8033    \ifnum\bbl@tempc<\bbl@tempb
8034       \edef\bbl@tempa{\fp_eval:n{\bbl@tempa-1}}% go back 1 year and redo
8035       \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
8036       \ifin@\def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
8037       \edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}%
8038    \fi
8039    \edef#4{\fp_eval:n{\bbl@tempa-621}}% set Jalali year
8040    \edef#6{\fp_eval:n{\bbl@tempc-\bbl@tempb+1}}% days from 1 farvardin
8041    \edef#5{\fp_eval:n{% set Jalali month
8042       (#6 <= 186) ? ceil(#6 / 31) : ceil((#6 - 6) / 30)}}
8043    \edef#6{\fp_eval:n{% set Jalali day
8044       (#6 - ((#5 <= 7) ? ((#5 - 1) * 31) : (((#5 - 1) * 30) + 6)))}}}}
8045 \ExplSyntaxOff
8046 ⟨/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.

```
8047 ⟨∗ca-coptic⟩
8048 \ExplSyntaxOn
8049 ⟨⟨Compute Julian day⟩⟩
8050 \def\bbl@ca@coptic#1-#2-#3\@@#4#5#6{%
8051   \edef\bbl@tempd{\fp_eval:n{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
8052   \edef\bbl@tempc{\fp_eval:n{\bbl@tempd - 1825029.5}}%
8053   \edef#4{\fp_eval:n{%
8054     floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
8055   \edef\bbl@tempc{\fp_eval:n{%
8056     \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1825029.5}}%
8057   \edef#5{\fp_eval:n{floor(\bbl@tempc / 30) + 1}}%
8058   \edef#6{\fp_eval:n{\bbl@tempc - (#5 - 1) * 30 + 1}}}
8059 \ExplSyntaxOff
8060 ⟨/ca-coptic⟩
8061 ⟨∗ca-ethiopic⟩
8062 \ExplSyntaxOn
8063 ⟨⟨Compute Julian day⟩⟩
8064 \def\bbl@ca@ethiopic#1-#2-#3\@@#4#5#6{%
8065   \edef\bbl@tempd{\fp_eval:n{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
8066   \edef\bbl@tempc{\fp_eval:n{\bbl@tempd - 1724220.5}}%
8067   \edef#4{\fp_eval:n{%
8068     floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
8069   \edef\bbl@tempc{\fp_eval:n{%
8070     \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1724220.5}}%
8071   \edef#5{\fp_eval:n{floor(\bbl@tempc / 30) + 1}}%
8072   \edef#6{\fp_eval:n{\bbl@tempc - (#5 - 1) * 30 + 1}}}
8073 \ExplSyntaxOff
8074 ⟨/ca-ethiopic⟩
```

19 Buddhist

That's very simple.

```
8075 ⟨∗ca-buddhist⟩
8076 \def\bbl@ca@buddhist#1-#2-#3\@@#4#5#6{%
8077   \edef#4{\number\numexpr#1+543\relax}%
8078   \edef#5{#2}%
8079   \edef#6{#3}}
8080 ⟨/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.

```
8081 ⟨∗bplain | blplain⟩
8082 \catcode`\{=1 % left brace is begin-group character
8083 \catcode`\}=2 % right brace is end-group character
8084 \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).

```
8085 \openin 0 hyphen.cfg
8086 \ifeof0
8087 \else
8088   \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.

```
8089   \def\input #1 {%
8090     \let\input\a
8091     \a hyphen.cfg
8092     \let\a\undefined
8093   }
8094 \fi
8095 ⟨/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.

```
8096 ⟨bplain⟩\a plain.tex
8097 ⟨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.

```
8098 ⟨bplain⟩\def\fmtname{babel-plain}
8099 ⟨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$_\varepsilon$ 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).

```
8100 ⟨⟨∗Emulate LaTeX⟩⟩ ≡
8101 \def\@empty{}
8102 \def\loadlocalcfg#1{%
8103   \openin0#1.cfg
8104   \ifeof0
8105     \closein0
8106   \else
8107     \closein0
8108     {\immediate\write16{***********************************}%
8109     \immediate\write16{* Local config file #1.cfg used}%
8110     \immediate\write16{*}%
8111     }
8112     \input #1.cfg\relax
8113   \fi
8114   \@endofldf}
```

20.3 General tools

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

```
8115 \long\def\@firstofone#1{#1}
8116 \long\def\@firstoftwo#1#2{#1}
8117 \long\def\@secondoftwo#1#2{#2}
8118 \def\@nnil{\@nil}
8119 \def\@gobbletwo#1#2{}
8120 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
8121 \def\@star@or@long#1{%
8122   \@ifstar
8123   {\let\l@ngrel@x\relax#1}%
8124   {\let\l@ngrel@x\long#1}}
8125 \let\l@ngrel@x\relax
8126 \def\@car#1#2\@nil{#1}
8127 \def\@cdr#1#2\@nil{#2}
8128 \let\@typeset@protect\relax
8129 \let\protected@edef\edef
8130 \long\def\@gobble#1{}
8131 \edef\@backslashchar{\expandafter\@gobble\string\\}
8132 \def\strip@prefix#1>{}
8133 \def\g@addto@macro#1#2{{%
8134     \toks@\expandafter{#1#2}%
8135     \xdef#1{\the\toks@}}}
8136 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
8137 \def\@nameuse#1{\csname #1\endcsname}
8138 \def\@ifundefined#1{%
8139   \expandafter\ifx\csname#1\endcsname\relax
8140     \expandafter\@firstoftwo
8141   \else
8142     \expandafter\@secondoftwo
8143   \fi}
8144 \def\@expandtwoargs#1#2#3{%
8145   \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
8146 \def\zap@space#1 #2{%
8147   #1%
8148   \ifx#2\@empty\else\expandafter\zap@space\fi
8149   #2}
8150 \let\bbl@trace\@gobble
8151 \def\bbl@error#1#2{%
8152   \begingroup
8153     \newlinechar=`\^^J
8154     \def\\{^^J(babel) }%
8155     \errhelp{#2}\errmessage{\\#1}%
8156   \endgroup}
8157 \def\bbl@warning#1{%
8158   \begingroup
8159     \newlinechar=`\^^J
8160     \def\\{^^J(babel) }%
8161     \message{\\#1}%
8162   \endgroup}
8163 \let\bbl@infowarn\bbl@warning
8164 \def\bbl@info#1{%
8165   \begingroup
8166     \newlinechar=`\^^J
8167     \def\\{^^J}%
8168     \wlog{#1}%
8169   \endgroup}
```

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

```
8170 \ifx\@preamblecmds\@undefined
8171   \def\@preamblecmds{}
```

```
8172 \fi
8173 \def\@onlypreamble#1{%
8174   \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
8175     \@preamblecmds\do#1}}
8176 \@onlypreamble\@onlypreamble
```

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

```
8177 \def\begindocument{%
8178   \@begindocumenthook
8179   \global\let\@begindocumenthook\@undefined
8180   \def\do##1{\global\let##1\@undefined}%
8181   \@preamblecmds
8182   \global\let\do\noexpand}
8183 \ifx\@begindocumenthook\@undefined
8184   \def\@begindocumenthook{}
8185 \fi
8186 \@onlypreamble\@begindocumenthook
8187 \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.

```
8188 \def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
8189 \@onlypreamble\AtEndOfPackage
8190 \def\@endofldf{}
8191 \@onlypreamble\@endofldf
8192 \let\bbl@afterlang\@empty
8193 \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.

```
8194 \catcode`\&=\z@
8195 \ifx&if@filesw\@undefined
8196   \expandafter\let\csname if@filesw\expandafter\endcsname
8197     \csname iffalse\endcsname
8198 \fi
8199 \catcode`\&=4
```

Mimick LaTeX's commands to define control sequences.

```
8200 \def\newcommand{\@star@or@long\new@command}
8201 \def\new@command#1{%
8202   \@testopt{\@newcommand#1}0}
8203 \def\@newcommand#1[#2]{%
8204   \@ifnextchar [{\@xargdef#1[#2]}%
8205                 {\@argdef#1[#2]}}
8206 \long\def\@argdef#1[#2]#3{%
8207   \@yargdef#1\@ne{#2}{#3}}
8208 \long\def\@xargdef#1[#2][#3]#4{%
8209   \expandafter\def\expandafter#1\expandafter{%
8210     \expandafter\@protected@testopt\expandafter #1%
8211     \csname\string#1\expandafter\endcsname{#3}}%
8212   \expandafter\@yargdef \csname\string#1\endcsname
8213   \tw@{#2}{#4}}
8214 \long\def\@yargdef#1#2#3{%
8215   \@tempcnta#3\relax
8216   \advance \@tempcnta \@ne
8217   \let\@hash@\relax
8218   \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
8219   \@tempcntb #2%
8220   \@whilenum\@tempcntb <\@tempcnta
8221   \do{%
8222     \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
8223     \advance\@tempcntb \@ne}%
```

```
8224    \let\@hash@##%
8225    \l@ngrel@x\expandafter\def\expandafter#1\reserved@a}
8226 \def\providecommand{\@star@or@long\provide@command}
8227 \def\provide@command#1{%
8228    \begingroup
8229       \escapechar\m@ne\xdef\@gtempa{{\string#1}}%
8230    \endgroup
8231    \expandafter\@ifundefined\@gtempa
8232       {\def\reserved@a{\new@command#1}}%
8233       {\let\reserved@a\relax
8234        \def\reserved@a{\new@command\reserved@a}}%
8235    \reserved@a}%

8236 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
8237 \def\declare@robustcommand#1{%
8238    \edef\reserved@a{\string#1}%
8239    \def\reserved@b{#1}%
8240    \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
8241    \edef#1{%
8242       \ifx\reserved@a\reserved@b
8243          \noexpand\x@protect
8244          \noexpand#1%
8245       \fi
8246       \noexpand\protect
8247       \expandafter\noexpand\csname
8248          \expandafter\@gobble\string#1 \endcsname
8249    }%
8250    \expandafter\new@command\csname
8251       \expandafter\@gobble\string#1 \endcsname
8252 }
8253 \def\x@protect#1{%
8254    \ifx\protect\@typeset@protect\else
8255       \@x@protect#1%
8256    \fi
8257 }
8258 \catcode`\&=\z@  % Trick to hide conditionals
8259    \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.

```
8260    \def\bbl@tempa{\csname newif\endcsname&ifin@}
8261 \catcode`\&=4
8262 \ifx\in@\@undefined
8263    \def\in@#1#2{%
8264       \def\in@@##1#1##2##3\in@@{%
8265          \ifx\in@##2\in@false\else\in@true\fi}%
8266       \in@@#2#1\in@\in@@}
8267 \else
8268    \let\bbl@tempa\@empty
8269 \fi
8270 \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).

```
8271 \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.

```
8272 \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.

```
8273 \ifx\@tempcnta\@undefined
8274   \csname newcount\endcsname\@tempcnta\relax
8275 \fi
8276 \ifx\@tempcntb\@undefined
8277   \csname newcount\endcsname\@tempcntb\relax
8278 \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).

```
8279 \ifx\bye\@undefined
8280   \advance\count10 by -2\relax
8281 \fi
8282 \ifx\@ifnextchar\@undefined
8283   \def\@ifnextchar#1#2#3{%
8284     \let\reserved@d=#1%
8285     \def\reserved@a{#2}\def\reserved@b{#3}%
8286     \futurelet\@let@token\@ifnch}
8287   \def\@ifnch{%
8288     \ifx\@let@token\@sptoken
8289       \let\reserved@c\@xifnch
8290     \else
8291       \ifx\@let@token\reserved@d
8292         \let\reserved@c\reserved@a
8293       \else
8294         \let\reserved@c\reserved@b
8295       \fi
8296     \fi
8297     \reserved@c}
8298   \def\:{\let\@sptoken= } \:  % this makes \@sptoken a space token
8299   \def\:{\@xifnch} \expandafter\def\: {\futurelet\@let@token\@ifnch}
8300 \fi
8301 \def\@testopt#1#2{%
8302   \@ifnextchar[{#1}{#1[#2]}}
8303 \def\@protected@testopt#1{%
8304   \ifx\protect\@typeset@protect
8305     \expandafter\@testopt
8306   \else
8307     \@x@protect#1%
8308   \fi}
8309 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
8310     #2\relax}\fi}
8311 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
8312         \else\expandafter\@gobble\fi{#1}}
```

20.4 Encoding related macros

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

```
8313 \def\DeclareTextCommand{%
8314   \@dec@text@cmd\providecommand
8315 }
8316 \def\ProvideTextCommand{%
8317   \@dec@text@cmd\providecommand
8318 }
8319 \def\DeclareTextSymbol#1#2#3{%
8320   \@dec@text@cmd\chardef#1{#2}#3\relax
8321 }
8322 \def\@dec@text@cmd#1#2#3{%
8323   \expandafter\def\expandafter#2%
8324       \expandafter{%
```

```
8325        \csname#3-cmd\expandafter\endcsname
8326        \expandafter#2%
8327        \csname#3\string#2\endcsname
8328      }%
8329 %   \let\@ifdefinable\@rc@ifdefinable
8330    \expandafter#1\csname#3\string#2\endcsname
8331 }
8332 \def\@current@cmd#1{%
8333    \ifx\protect\@typeset@protect\else
8334        \noexpand#1\expandafter\@gobble
8335    \fi
8336 }
8337 \def\@changed@cmd#1#2{%
8338    \ifx\protect\@typeset@protect
8339        \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
8340            \expandafter\ifx\csname ?\string#1\endcsname\relax
8341                \expandafter\def\csname ?\string#1\endcsname{%
8342                    \@changed@x@err{#1}%
8343                }%
8344            \fi
8345            \global\expandafter\let
8346                \csname\cf@encoding \string#1\expandafter\endcsname
8347                \csname ?\string#1\endcsname
8348        \fi
8349        \csname\cf@encoding\string#1%
8350            \expandafter\endcsname
8351    \else
8352        \noexpand#1%
8353    \fi
8354 }
8355 \def\@changed@x@err#1{%
8356    \errhelp{Your command will be ignored, type <return> to proceed}%
8357    \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
8358 \def\DeclareTextCommandDefault#1{%
8359    \DeclareTextCommand#1?%
8360 }
8361 \def\ProvideTextCommandDefault#1{%
8362    \ProvideTextCommand#1?%
8363 }
8364 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
8365 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
8366 \def\DeclareTextAccent#1#2#3{%
8367   \DeclareTextCommand#1{#2}[1]{\accent#3 ##1}
8368 }
8369 \def\DeclareTextCompositeCommand#1#2#3#4{%
8370    \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
8371    \edef\reserved@b{\string##1}%
8372    \edef\reserved@c{%
8373      \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
8374    \ifx\reserved@b\reserved@c
8375      \expandafter\expandafter\expandafter\ifx
8376          \expandafter\@car\reserved@a\relax\relax\@nil
8377          \@text@composite
8378      \else
8379        \edef\reserved@b##1{%
8380            \def\expandafter\noexpand
8381                \csname#2\string#1\endcsname####1{%
8382                \noexpand\@text@composite
8383                    \expandafter\noexpand\csname#2\string#1\endcsname
8384                    ####1\noexpand\@empty\noexpand\@text@composite
8385                    {##1}%
8386            }%
8387        }%
```

224

```
8388          \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
8389        \fi
8390        \expandafter\def\csname\expandafter\string\csname
8391            #2\endcsname\string#1-\string#3\endcsname{#4}
8392      \else
8393        \errhelp{Your command will be ignored, type <return> to proceed}%
8394        \errmessage{\string\DeclareTextCompositeCommand\space used on
8395            inappropriate command \protect#1}
8396      \fi
8397 }
8398 \def\@text@composite#1#2#3\@text@composite{%
8399      \expandafter\@text@composite@x
8400        \csname\string#1-\string#2\endcsname
8401 }
8402 \def\@text@composite@x#1#2{%
8403      \ifx#1\relax
8404          #2%
8405      \else
8406          #1%
8407      \fi
8408 }
8409 %
8410 \def\@strip@args#1:#2-#3\@strip@args{#2}
8411 \def\DeclareTextComposite#1#2#3#4{%
8412      \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
8413      \bgroup
8414          \lccode`\@=#4%
8415          \lowercase{%
8416      \egroup
8417          \reserved@a @%
8418      }%
8419 }
8420 %
8421 \def\UseTextSymbol#1#2{#2}
8422 \def\UseTextAccent#1#2#3{}
8423 \def\@use@text@encoding#1{}
8424 \def\DeclareTextSymbolDefault#1#2{%
8425      \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
8426 }
8427 \def\DeclareTextAccentDefault#1#2{%
8428      \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
8429 }
8430 \def\cf@encoding{OT1}
```

Currently we only use the LaTeX2ε method for accents for those that are known to be made active in *some* language definition file.

```
8431 \DeclareTextAccent{\"}{OT1}{127}
8432 \DeclareTextAccent{\'}{OT1}{19}
8433 \DeclareTextAccent{\^}{OT1}{94}
8434 \DeclareTextAccent{\`}{OT1}{18}
8435 \DeclareTextAccent{\~}{OT1}{126}
```

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

```
8436 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
8437 \DeclareTextSymbol{\textquotedblright}{OT1}{`\"}
8438 \DeclareTextSymbol{\textquoteleft}{OT1}{`\`}
8439 \DeclareTextSymbol{\textquoteright}{OT1}{`\'}
8440 \DeclareTextSymbol{\i}{OT1}{16}
8441 \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.

```
8442 \ifx\scriptsize\@undefined
8443    \let\scriptsize\sevenrm
```

225

```
8444 \fi
```

And a few more "dummy" definitions.

```
8445 \def\languagename{english}%
8446 \let\bbl@opt@shorthands\@nnil
8447 \def\bbl@ifshorthand#1#2#3{#2}%
8448 \let\bbl@language@opts\@empty
8449 \ifx\babeloptionstrings\@undefined
8450   \let\bbl@opt@strings\@nnil
8451 \else
8452   \let\bbl@opt@strings\babeloptionstrings
8453 \fi
8454 \def\BabelStringsDefault{generic}
8455 \def\bbl@tempa{normal}
8456 \ifx\babeloptionmath\bbl@tempa
8457   \def\bbl@mathnormal{\noexpand\textormath}
8458 \fi
8459 \def\AfterBabelLanguage#1#2{}
8460 \ifx\BabelModifiers\@undefined\let\BabelModifiers\relax\fi
8461 \let\bbl@afterlang\relax
8462 \def\bbl@opt@safe{BR}
8463 \ifx\@uclclist\@undefined\let\@uclclist\@empty\fi
8464 \ifx\bbl@trace\@undefined\def\bbl@trace#1{}\fi
8465 \expandafter\newif\csname ifbbl@single\endcsname
8466 \chardef\bbl@bidimode\z@
8467 ⟨⟨/Emulate LaTeX⟩⟩
```

A proxy file:

```
8468 ⟨∗plain⟩
8469 \input babel.def
8470 ⟨/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).