Babel

Javier Bezos
Current maintainer

Johannes L. Braams
Original author

Localization and internationalization

Unicode

T$_E$X

pdfT$_E$X

LuaT$_E$X

XeT$_E$X

Contents

Troubleshoooting

3

Part I

User guide

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

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

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

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

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

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

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

1   The user interface

1.1   Monolingual documents

In most cases, a single language is required, and then all you need in LaTeX is to load the package using its standard mechanism for this purpose, namely, passing that language as an optional argument. In addition, you may want to set the font and input encodings. Another approach is making the language a global option in order to let other packages detect and use it. This is the standard way in LaTeX for an option – in this case a language – to be recognized by several packages.
Many languages are compatible with xetex and luatex. With them you can use babel to localize the documents. When these engines are used, the Latin script is covered by default in current LaTeX (provided the document encoding is UTF-8), because the font loader is preloaded and the font is switched to `lmroman`. Other scripts require loading fontspec. You may want to set the font attributes with fontspec, too.

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

```
PDFTEX
        \documentclass{article}

        \usepackage[T1]{fontenc}
```

```
\usepackage[french]{babel}

\begin{document}

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

\end{document}
```

Now consider something like:

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

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

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

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

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, \aliashorthands is ignored).

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

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

WARNING   Shorthands remember somehow the original character, and the fallback value is that of the latter. So, in this example, if no shorthand if found, ^ expands to a non-breaking space, because this is the value of ~ (internally, ^ still calls \active@char~ or \normal@char~). Furthermore, if you change the system value of ^ with \defineshorthand nothing happens.

1.11   Package options

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

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

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

activegrave   Same for `.

shorthands=   ⟨*char*⟩⟨*char*⟩… | off

The only language shorthands activated are those given, like, eg:

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

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

---

[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.

<small>LUATEX/XETEX</small>

```
\documentclass{book}

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

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

\begin{document}

\tableofcontents

\chapter{სამზარეულო და სუფრის ტრადიციები}

ქართული ტრადიციული სამზარეულო ერთ-ერთი უძიდრესია მთეჲ მსოფლიოში.

\end{document}
```

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

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

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

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

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

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

Or also:

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

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

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

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

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

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

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

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

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

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

Code	Language	Code	Language
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

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]

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

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

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

chinese-hans-sg

chinese-hans

chinese-hant-hk

chinese-hant-mo

chinese-hant

chinese-simplified-hongkongsarchina

chinese-simplified-macausarchina

chinese-simplified-singapore

chinese-simplified

chinese-traditional-hongkongsarchina

chinese-traditional-macausarchina

chinese-traditional

chinese

churchslavic

churchslavic-cyrs

churchslavic-oldcyrillic[12]

churchsslavic-glag

churchsslavic-glagolitic

colognian

cornish

croatian

czech

danish

duala

dutch

dzongkha

embu

english-au

english-australia

english-ca

english-canada

english-gb

english-newzealand

english-nz

english-unitedkingdom

english-unitedstates

english-us

english

esperanto

estonian

ewe

ewondo

faroese

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*⟩}
\localecounterl {⟨*style*⟩}{⟨*counter*⟩}

New 3.41 Many 'ini' locale files has been extended with information about non-positional numerical systems, based on those predefined in CSS. They only work with xetex and luatex and are fully expendable (even inside an unprotected \edef). Currently, they are limited to numbers below 10000.

There are several ways to use them (for the availabe styles in each language, see the list below):

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

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

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

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

The styles are:

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

36

\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 ([ïü]), the replacement could be {1|ïü|íú}, which maps *ï* to *í*, and *ü* to *ú*, so that the diaeresis is removed.

This feature is activated with the first `\babelposthyphenation` or `\babelprehyphenation`.
New 3.85 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:

```
\babelprehenation[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

41

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` `\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.

\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=3.86.07774⟩⟩
2 ⟨⟨date=2023/03/18⟩⟩
```

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
\bbl@afterfi Because the code that is used in the handling of active characters may need to look ahead, we take 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 The stack is simply a list of languagenames, separated with a '+' sign; the push function can be simple:
\bbl@pop@language

```
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#1#2{%
1013   \ifx\UseHook\@undefined\else\UseHook{babel/*/#1}\fi
1014   \def\bbl@elth##1{%
1015     \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#1@}#2}}%
1016   \bbl@cs{ev@#1@}%
1017   \ifx\languagename\@undefined\else % Test required for Plain (?)
1018     \ifx\UseHook\@undefined\else\UseHook{babel/\languagename/#1}\fi
1019     \def\bbl@elth##1{%
1020       \bbl@cs{hk@##1}{\bbl@cl{ev@##1@#1}#2}}%
1021     \bbl@cl{ev@#1}%
1022   \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).

```
1023 \def\bbl@evargs{,% <- don't delete this comma
1024   everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
1025   adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
1026   beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
1027   hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%
1028   beforestart=0,languagename=2}
1029 \ifx\NewHook\@undefined\else
1030   \def\bbl@tempa#1=#2\@@{\NewHook{babel/#1}}
1031   \bbl@foreach\bbl@evargs{\bbl@tempa#1\@@}
1032 \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.

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

```
1073                ########1}}}}%
1074          {}%
1075        \toks@\expandafter{##1}%
1076        \edef##1{%
1077          \bbl@csarg\noexpand{ensure@\languagename}%
1078          {\the\toks@}}%
1079      \fi
1080      \expandafter\bbl@tempb
1081    \fi}%
1082  \expandafter\bbl@tempb\bbl@captionslist\today\@empty
1083  \def\bbl@tempa##1{% elt for include list
1084    \ifx##1\@empty\else
1085      \bbl@csarg\in@{ensure@\languagename\expandafter}\expandafter{##1}%
1086      \ifin@\else
1087        \bbl@tempb##1\@empty
1088      \fi
1089      \expandafter\bbl@tempa
1090    \fi}%
1091  \bbl@tempa#1\@empty}
1092 \def\bbl@captionslist{%
1093   \prefacename\refname\abstractname\bibname\chaptername\appendixname
1094   \contentsname\listfigurename\listtablename\indexname\figurename
1095   \tablename\partname\enclname\ccname\headtoname\pagename\seename
1096   \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.

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

```
1115    \ifx#2\@undefined\else
1116      \ldf@quit{#1}%
1117    \fi
1118    \else
1119    \expandafter\ifx\csname#2\endcsname\relax\else
1120      \ldf@quit{#1}%
1121    \fi
1122    \fi
1123    \bbl@ldfinit}
```

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

```
1124 \def\ldf@quit#1{%
1125    \expandafter\main@language\expandafter{#1}%
1126    \catcode`\@=\atcatcode \let\atcatcode\relax
1127    \catcode`\==\eqcatcode \let\eqcatcode\relax
1128    \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.

```
1129 \def\bbl@afterldf#1{% TODO. Merge into the next macro? Unused elsewhere
1130    \bbl@afterlang
1131    \let\bbl@afterlang\relax
1132    \let\BabelModifiers\relax
1133    \let\bbl@screset\relax}%
1134 \def\ldf@finish#1{%
1135    \loadlocalcfg{#1}%
1136    \bbl@afterldf{#1}%
1137    \expandafter\main@language\expandafter{#1}%
1138    \catcode`\@=\atcatcode \let\atcatcode\relax
1139    \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.

```
1140 \@onlypreamble\LdfInit
1141 \@onlypreamble\ldf@quit
1142 \@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.

```
1143 \def\main@language#1{%
1144    \def\bbl@main@language{#1}%
1145    \let\languagename\bbl@main@language % TODO. Set localename
1146    \bbl@id@assign
1147    \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.

```
1148 \def\bbl@beforestart{%
1149    \def\@nolanerr##1{%
1150      \bbl@warning{Undefined language '##1' in aux.\\Reported}}%
1151    \bbl@usehooks{beforestart}{}%
1152    \global\let\bbl@beforestart\relax}
1153 \AtBeginDocument{%
1154    {\@nameuse{bbl@beforestart}}%  Group!
1155    \if@filesw
1156      \providecommand\babel@aux[2]{}%
1157      \immediate\write\@mainaux{%
1158        \string\providecommand\string\babel@aux[2]{}}%
```

```
1159        \immediate\write\@mainaux{\string\@nameuse{bbl@beforestart}}}%
1160    \fi
1161    \expandafter\selectlanguage\expandafter{\bbl@main@language}%
1162    \ifbbl@single  % must go after the line above.
1163        \renewcommand\selectlanguage[1]{}%
1164        \renewcommand\foreignlanguage[2]{#2}%
1165        \global\let\babel@aux\@gobbletwo  % Also as flag
1166    \fi
1167    \ifcase\bbl@engine\or\pagedir\bodydir\fi} % TODO - a better place
```

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

```
1168 \def\select@language@x#1{%
1169    \ifcase\bbl@select@type
1170        \bbl@ifsamestring\languagename{#1}{}{\select@language{#1}}%
1171    \else
1172        \select@language{#1}%
1173    \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.

```
1174 \bbl@trace{Shorhands}
1175 \def\bbl@add@special#1{% 1:a macro like \", \?, etc.
1176    \bbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
1177    \bbl@ifunset{@sanitize}{}{\bbl@add\@sanitize{\@makeother#1}}%
1178    \ifx\nfss@catcodes\@undefined\else % TODO - same for above
1179        \begingroup
1180            \catcode`#1\active
1181            \nfss@catcodes
1182            \ifnum\catcode`#1=\active
1183                \endgroup
1184                \bbl@add\nfss@catcodes{\@makeother#1}%
1185            \else
1186                \endgroup
1187            \fi
1188    \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.

```
1189 \def\bbl@remove@special#1{%
1190    \begingroup
1191        \def\x##1##2{\ifnum`#1=`##2\noexpand\@empty
1192                    \else\noexpand##1\noexpand##2\fi}%
1193        \def\do{\x\do}%
1194        \def\@makeother{\x\@makeother}%
1195    \edef\x{\endgroup
1196        \def\noexpand\dospecials{\dospecials}%
1197        \expandafter\ifx\csname @sanitize\endcsname\relax\else
1198            \def\noexpand\@sanitize{\@sanitize}%
1199        \fi}%
1200    \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`).

```
1201 \def\bbl@active@def#1#2#3#4{%
1202   \@namedef{#3#1}{%
1203     \expandafter\ifx\csname#2@sh@#1@\endcsname\relax
1204       \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
1205     \else
1206       \bbl@afterfi\csname#2@sh@#1@\endcsname
1207   \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.

```
1208   \long\@namedef{#3@arg#1}##1{%
1209     \expandafter\ifx\csname#2@sh@#1@\string##1@\endcsname\relax
1210       \bbl@afterelse\csname#4#1\endcsname##1%
1211     \else
1212       \bbl@afterfi\csname#2@sh@#1@\string##1@\endcsname
1213   \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.

```
1214 \def\initiate@active@char#1{%
1215   \bbl@ifunset{active@char\string#1}%
1216     {\bbl@withactive
1217       {\expandafter\@initiate@active@char\expandafter}#1\string#1#1}%
1218     {}}
```

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).

```
1219 \def\@initiate@active@char#1#2#3{%
1220   \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
1221   \ifx#1\@undefined
1222     \bbl@csarg\def{oridef@#2}{\def#1{\active@prefix#1\@undefined}}%
1223   \else
1224     \bbl@csarg\let{oridef@@#2}#1%
1225     \bbl@csarg\edef{oridef@#2}{%
1226       \let\noexpand#1%
1227       \expandafter\noexpand\csname bbl@oridef@@#2\endcsname}%
1228   \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*).

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

```
1235        \textormath{#3}{\csname bbl@oridef@@#2\endcsname}}%
1236    \else
1237      \@namedef{normal@char#2}{#3}%
1238    \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).

```
1239    \bbl@restoreactive{#2}%
1240    \AtBeginDocument{%
1241      \catcode`#2\active
1242      \if@filesw
1243        \immediate\write\@mainaux{\catcode`\string#2\active}%
1244      \fi}%
1245    \expandafter\bbl@add@special\csname#2\endcsname
1246    \catcode`#2\active
1247  \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*⟩).

```
1248    \let\bbl@tempa\@firstoftwo
1249    \if\string^#2%
1250      \def\bbl@tempa{\noexpand\textormath}%
1251    \else
1252      \ifx\bbl@mathnormal\@undefined\else
1253        \let\bbl@tempa\bbl@mathnormal
1254      \fi
1255    \fi
1256    \expandafter\edef\csname active@char#2\endcsname{%
1257      \bbl@tempa
1258        {\noexpand\if@safe@actives
1259          \noexpand\expandafter
1260          \expandafter\noexpand\csname normal@char#2\endcsname
1261        \noexpand\else
1262          \noexpand\expandafter
1263          \expandafter\noexpand\csname bbl@doactive#2\endcsname
1264        \noexpand\fi}%
1265      {\expandafter\noexpand\csname normal@char#2\endcsname}}%
1266    \bbl@csarg\edef{doactive#2}{%
1267      \expandafter\noexpand\csname user@active#2\endcsname}%
```

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

$$\text{\active@prefix } ⟨char⟩ \text{ \normal@char}⟨char⟩$$

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

```
1268    \bbl@csarg\edef{active@#2}{%
1269      \noexpand\active@prefix\noexpand#1%
1270      \expandafter\noexpand\csname active@char#2\endcsname}%
1271    \bbl@csarg\edef{normal@#2}{%
1272      \noexpand\active@prefix\noexpand#1%
1273      \expandafter\noexpand\csname normal@char#2\endcsname}%
1274    \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.

```
1275    \bbl@active@def#2\user@group{user@active}{language@active}%
1276    \bbl@active@def#2\language@group{language@active}{system@active}%
1277    \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.

```
1278    \expandafter\edef\csname\user@group @sh@#2@@\endcsname
1279       {\expandafter\noexpand\csname normal@char#2\endcsname}%
1280    \expandafter\edef\csname\user@group @sh@#2@\string\protect@\endcsname
1281       {\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.

```
1282    \if\string'#2%
1283       \let\prim@s\bbl@prim@s
1284       \let\active@math@prime#1%
1285    \fi
1286    \bbl@usehooks{initiateactive}{{#1}{#2}{#3}}}
```

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

```
1287 ⟨⟨*More package options⟩⟩ ≡
1288 \DeclareOption{math=active}{}
1289 \DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
1290 ⟨⟨/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`.

```
1291 \@ifpackagewith{babel}{KeepShorthandsActive}%
1292    {\let\bbl@restoreactive\@gobble}%
1293    {\def\bbl@restoreactive#1{%
1294       \bbl@exp{%
1295          \\\AfterBabelLanguage\\\CurrentOption
1296             {\catcode`#1=\the\catcode`#1\relax}%
1297          \\\AtEndOfPackage
1298             {\catcode`#1=\the\catcode`#1\relax}}}%
1299    \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.

```
1300 \def\bbl@sh@select#1#2{%
1301    \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
1302       \bbl@afterelse\bbl@scndcs
1303    \else
1304       \bbl@afterfi\csname#1@sh@#2@sel\endcsname
1305    \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 `\protect`s 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.

```
1306 \begingroup
```

91

```
1307 \bbl@ifunset{ifincsname}% TODO. Ugly. Correct? Only Plain?
1308   {\gdef\active@prefix#1{%
1309     \ifx\protect\@typeset@protect
1310     \else
1311       \ifx\protect\@unexpandable@protect
1312         \noexpand#1%
1313       \else
1314         \protect#1%
1315       \fi
1316       \expandafter\@gobble
1317     \fi}}
1318   {\gdef\active@prefix#1{%
1319     \ifincsname
1320       \string#1%
1321       \expandafter\@gobble
1322     \else
1323       \ifx\protect\@typeset@protect
1324       \else
1325         \ifx\protect\@unexpandable@protect
1326           \noexpand#1%
1327         \else
1328           \protect#1%
1329         \fi
1330         \expandafter\expandafter\expandafter\@gobble
1331       \fi
1332     \fi}}
1333 \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⟨*char*⟩. 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).

```
1334 \newif\if@safe@actives
1335 \@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.

```
1336 \def\bbl@restore@actives{\if@safe@actives\@safe@activesfalse\fi}
```

\bbl@activate Both macros take one argument, like \initiate@active@char. The macro is used to change the
\bbl@deactivate definition of an active character to expand to \active@char⟨*char*⟩ in the case of \bbl@activate, or \normal@char⟨*char*⟩ in the case of \bbl@deactivate.

```
1337 \chardef\bbl@activated\z@
1338 \def\bbl@activate#1{%
1339   \chardef\bbl@activated\@ne
1340   \bbl@withactive{\expandafter\let\expandafter}#1%
1341     \csname bbl@active@\string#1\endcsname}
1342 \def\bbl@deactivate#1{%
1343   \chardef\bbl@activated\tw@
1344   \bbl@withactive{\expandafter\let\expandafter}#1%
1345     \csname bbl@normal@\string#1\endcsname}
```

\bbl@firstcs These macros are used only as a trick when declaring shorthands.
\bbl@scndcs
```
1346 \def\bbl@firstcs#1#2{\csname#1\endcsname}
1347 \def\bbl@scndcs#1#2{\csname#2\endcsname}
```

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

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

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

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

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

```
1348 \def\babel@texpdf#1#2#3#4{%
1349   \ifx\texorpdfstring\@undefined
1350     \textormath{#1}{#3}%
1351   \else
1352     \texorpdfstring{\textormath{#1}{#3}}{#2}%
1353     % \texorpdfstring{\textormath{#1}{#3}}{\textormath{#2}{#4}}%
1354   \fi}
1355 %
1356 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
1357 \def\@decl@short#1#2#3\@nil#4{%
1358   \def\bbl@tempa{#3}%
1359   \ifx\bbl@tempa\@empty
1360     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
1361     \bbl@ifunset{#1@sh@\string#2@}{}%
1362       {\def\bbl@tempa{#4}%
1363        \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbl@tempa
1364        \else
1365          \bbl@info
1366            {Redefining #1 shorthand \string#2\\%
1367             in language \CurrentOption}%
1368        \fi}%
1369     \@namedef{#1@sh@\string#2@}{#4}%
1370   \else
1371     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs
1372     \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%
1373       {\def\bbl@tempa{#4}%
1374        \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
1375        \else
1376          \bbl@info
1377            {Redefining #1 shorthand \string#2\string#3\\%
1378             in language \CurrentOption}%
1379        \fi}%
1380     \@namedef{#1@sh@\string#2@\string#3@}{#4}%
1381   \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.

```
1382 \def\textormath{%
1383   \ifmmode
1384     \expandafter\@secondoftwo
1385   \else
1386     \expandafter\@firstoftwo
1387   \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'.

```
1388 \def\user@group{user}
1389 \def\language@group{english} % TODO. I don't like defaults
1390 \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.

```
1391 \def\useshorthands{%
1392   \@ifstar\bbl@usesh@s{\bbl@usesh@x{}}}
1393 \def\bbl@usesh@s#1{%
1394   \bbl@usesh@x
1395     {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bbl@activate{#1}}}%
1396     {#1}}
1397 \def\bbl@usesh@x#1#2{%
1398   \bbl@ifshorthand{#2}%
1399     {\def\user@group{user}%
1400      \initiate@active@char{#2}%
1401      #1%
1402      \bbl@activate{#2}}%
1403     {\bbl@error
1404       {I can't declare a shorthand turned off (\string#2)}
1405       {Sorry, but you can't use shorthands which have been\\%
1406        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.

```
1407 \def\user@language@group{user@\language@group}
1408 \def\bbl@set@user@generic#1#2{%
1409   \bbl@ifunset{user@generic@active#1}%
1410     {\bbl@active@def#1\user@language@group{user@active}{user@generic@active}%
1411      \bbl@active@def#1\user@group{user@generic@active}{language@active}%
1412      \expandafter\edef\csname#2@sh@#1@@\endcsname{%
1413        \expandafter\noexpand\csname normal@char#1\endcsname}%
1414      \expandafter\edef\csname#2@sh@#1@\string\protect@\endcsname{%
1415        \expandafter\noexpand\csname user@active#1\endcsname}}%
1416   \@empty}
1417 \newcommand\defineshorthand[3][user]{%
1418   \edef\bbl@tempa{\zap@space#1 \@empty}%
1419   \bbl@for\bbl@tempb\bbl@tempa{%
1420     \if*\expandafter\@car\bbl@tempb\@nil
1421       \edef\bbl@tempb{user@\expandafter\@gobble\bbl@tempb}%
1422       \@expandtwoargs
1423         \bbl@set@user@generic{\expandafter\string\@car#2\@nil}\bbl@tempb
1424     \fi
1425     \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].

```
1426 \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".

```
1427 \def\aliasshorthand#1#2{%
1428   \bbl@ifshorthand{#2}%
1429     {\expandafter\ifx\csname active@char\string#2\endcsname\relax
1430       \ifx\document\@notprerr
1431         \@notshorthand{#2}%
1432       \else
1433         \initiate@active@char{#2}%
1434         \bbl@ccarg\let{active@char\string#2}{active@char\string#1}%
1435         \bbl@ccarg\let{normal@char\string#2}{normal@char\string#1}%
1436         \bbl@activate{#2}%
1437       \fi
1438     \fi}%
1439     {\bbl@error
```

```
1440            {Cannot declare a shorthand turned off (\string#2)}
1441            {Sorry, but you cannot use shorthands which have been\\%
1442             turned off in the package options}}}
```

\@notshorthand

```
1443 \def\@notshorthand#1{%
1444   \bbl@error{%
1445     The character '\string #1' should be made a shorthand character;\\%
1446     add the command \string\useshorthands\string{#1\string} to
1447     the preamble.\\%
1448     I will ignore your instruction}%
1449   {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.

```
1450 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
1451 \DeclareRobustCommand*\shorthandoff{%
1452   \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
1453 \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.

```
1454 \def\bbl@switch@sh#1#2{%
1455   \ifx#2\@nnil\else
1456     \bbl@ifunset{bbl@active@\string#2}%
1457       {\bbl@error
1458         {I can't switch '\string#2' on or off--not a shorthand}%
1459         {This character is not a shorthand. Maybe you made\\%
1460          a typing mistake? I will ignore your instruction.}}%
1461       {\ifcase#1%   off, on, off*
1462         \catcode`#212\relax
1463        \or
1464         \catcode`#2\active
1465         \bbl@ifunset{bbl@shdef@\string#2}%
1466           {}%
1467           {\bbl@withactive{\expandafter\let\expandafter}#2%
1468              \csname bbl@shdef@\string#2\endcsname
1469            \bbl@csarg\let{shdef@\string#2}\relax}%
1470         \ifcase\bbl@activated\or
1471           \bbl@activate{#2}%
1472         \else
1473           \bbl@deactivate{#2}%
1474         \fi
1475        \or
1476         \bbl@ifunset{bbl@shdef@\string#2}%
1477           {\bbl@withactive{\bbl@csarg\let{shdef@\string#2}}#2}%
1478           {}%
1479         \csname bbl@oricat@\string#2\endcsname
1480         \csname bbl@oridef@\string#2\endcsname
1481       \fi}%
1482     \bbl@afterfi\bbl@switch@sh#1%
1483   \fi}
```

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

```
1484 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
1485 \def\bbl@putsh#1{%
1486   \bbl@ifunset{bbl@active@\string#1}%
1487     {\bbl@putsh@i#1\@empty\@nnil}%
```

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

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

```
1510 \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.

```
1511 \def\bbl@prim@s{%
1512   \prime\futurelet\@let@token\bbl@pr@m@s}
1513 \def\bbl@if@primes#1#2{%
1514   \ifx#1\@let@token
1515     \expandafter\@firstoftwo
1516   \else\ifx#2\@let@token
1517     \bbl@afterelse\expandafter\@firstoftwo
1518   \else
1519     \bbl@afterfi\expandafter\@secondoftwo
1520   \fi\fi}
1521 \begingroup
1522   \catcode`\^=7  \catcode`\*=\active  \lccode`\*=`\^
1523   \catcode`\'=12 \catcode`\"=\active  \lccode`\"=`\'
1524   \lowercase{%
1525     \gdef\bbl@pr@m@s{%
1526       \bbl@if@primes"'%
1527         \pr@@@s
1528         {\bbl@if@primes*^\pr@@@t\egroup}}}
1529 \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).

```
1530 \initiate@active@char{~}
1531 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
1532 \bbl@activate{~}
```

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

```
1533 \expandafter\def\csname OT1dqpos\endcsname{127}
1534 \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

```
1535 \ifx\f@encoding\@undefined
1536   \def\f@encoding{OT1}
1537 \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.

```
1538 \bbl@trace{Language attributes}
1539 \newcommand\languageattribute[2]{%
1540   \def\bbl@tempc{#1}%
1541   \bbl@fixname\bbl@tempc
1542   \bbl@iflanguage\bbl@tempc{%
1543     \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.

```
1544     \ifx\bbl@known@attribs\@undefined
1545       \in@false
1546     \else
1547       \bbl@xin@{,\bbl@tempc-##1,}{,\bbl@known@attribs,}%
1548     \fi
1549     \ifin@
1550       \bbl@warning{%
1551         You have more than once selected the attribute '##1'\\%
1552         for language #1. Reported}%
1553     \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.

```
1554       \bbl@exp{%
1555         \\\bbl@add@list\\\bbl@known@attribs{\bbl@tempc-##1}}%
1556       \edef\bbl@tempa{\bbl@tempc-##1}%
1557       \expandafter\bbl@ifknown@ttrib\expandafter{\bbl@tempa}\bbl@attributes%
1558       {\csname\bbl@tempc @attr@##1\endcsname}%
1559       {\@attrerr{\bbl@tempc}{##1}}%
1560     \fi}}}
1561 \@onlypreamble\languageattribute
```

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

```
1562 \newcommand*{\@attrerr}[2]{%
1563   \bbl@error
1564     {The attribute #2 is unknown for language #1.}%
1565     {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}.

```
1566 \def\bbl@declare@ttribute#1#2#3{%
1567   \bbl@xin@{,#2,}{,\BabelModifiers,}%
1568   \ifin@
1569     \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
1570   \fi
1571   \bbl@add@list\bbl@attributes{#1-#2}%
1572   \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.

```
1573 \def\bbl@ifattributeset#1#2#3#4{%
1574   \ifx\bbl@known@attribs\@undefined
1575     \in@false
1576   \else
1577     \bbl@xin@{,#1-#2,}{,\bbl@known@attribs,}%
1578   \fi
1579   \ifin@
1580     \bbl@afterelse#3%
1581   \else
1582     \bbl@afterfi#4%
1583   \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.

```
1584 \def\bbl@ifknown@ttrib#1#2{%
1585   \let\bbl@tempa\@secondoftwo
1586   \bbl@loopx\bbl@tempb{#2}{%
1587     \expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{,#1,}%
1588     \ifin@
1589       \let\bbl@tempa\@firstoftwo
1590     \else
1591     \fi}%
1592   \bbl@tempa}
```

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

```
1593 \def\bbl@clear@ttribs{%
1594   \ifx\bbl@attributes\@undefined\else
1595     \bbl@loopx\bbl@tempa{\bbl@attributes}{%
1596       \expandafter\bbl@clear@ttrib\bbl@tempa.
1597       }%
1598     \let\bbl@attributes\@undefined
1599   \fi}
1600 \def\bbl@clear@ttrib#1-#2.{%
1601   \expandafter\let\csname#1@attr@#2\endcsname\@undefined}
1602 \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
```
1603 \bbl@trace{Macros for saving definitions}
1604 \def\babel@beginsave{\babel@savecnt\z@}
```

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

```
1605 \newcount\babel@savecnt
1606 \babel@beginsave
```

\babel@save \
\babel@savevariable

The macro \babel@save⟨*csname*⟩ saves the current meaning of the control sequence ⟨*csname*⟩ to \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.

```
1607 \def\babel@save#1{%
1608   \def\bbl@tempa{{,#1,}}% Clumsy, for Plain
1609   \expandafter\bbl@add\expandafter\bbl@tempa\expandafter{%
1610     \expandafter{\expandafter,\bbl@savedextras,}}%
1611   \expandafter\in@\bbl@tempa
1612   \ifin@\else
1613     \bbl@add\bbl@savedextras{,#1,}%
1614     \bbl@carg\let{babel@\number\babel@savecnt}#1\relax
1615     \toks@\expandafter{\originalTeX\let#1=}%
1616     \bbl@exp{%
1617       \def\\\originalTeX{\the\toks@\<babel@\number\babel@savecnt>\relax}}%
1618     \advance\babel@savecnt\@ne
1619   \fi}
1620 \def\babel@savevariable#1{%
1621   \toks@\expandafter{\originalTeX #1=}%
1622   \bbl@exp{\def\\\originalTeX{\the\toks@\the#1\relax}}}
```

\bbl@frenchspacing \
\bbl@nonfrenchspacing

Some languages need to have \frenchspacing in effect. Others don't want that. The command \bbl@frenchspacing switches it on when it isn't already in effect and \bbl@nonfrenchspacing switches it off if necessary. A more refined way to switch the catcodes is done with ini files. Here an auxiliary macro is defined, but the main part is in \babelprovide. This new method should be ideally the default one.

```
1623 \def\bbl@frenchspacing{%
1624   \ifnum\the\sfcode`\.=\@m
1625     \let\bbl@nonfrenchspacing\relax
1626   \else
1627     \frenchspacing
1628     \let\bbl@nonfrenchspacing\nonfrenchspacing
1629   \fi}
1630 \let\bbl@nonfrenchspacing\nonfrenchspacing
1631 \let\bbl@elt\relax
1632 \edef\bbl@fs@chars{%
1633   \bbl@elt{\string.}\@m{3000}\bbl@elt{\string?}\@m{3000}%
1634   \bbl@elt{\string!}\@m{3000}\bbl@elt{\string:}\@m{2000}%
1635   \bbl@elt{\string;}\@m{1500}\bbl@elt{\string,}\@m{1250}}
1636 \def\bbl@pre@fs{%
1637   \def\bbl@elt##1##2##3{\sfcode`##1=\the\sfcode`##1\relax}%
1638   \edef\bbl@save@sfcodes{\bbl@fs@chars}}
1639 \def\bbl@post@fs{%
1640   \bbl@save@sfcodes
1641   \edef\bbl@tempa{\bbl@cl{frspc}}%
1642   \edef\bbl@tempa{\expandafter\@car\bbl@tempa\@nil}%
1643   \if u\bbl@tempa          % do nothing
1644   \else\if n\bbl@tempa      % non french
1645     \def\bbl@elt##1##2##3{%
1646       \ifnum\sfcode`##1=##2\relax
1647         \babel@savevariable{\sfcode`##1}%
1648         \sfcode`##1=##3\relax
1649       \fi}%
1650     \bbl@fs@chars
1651   \else\if y\bbl@tempa      % french
1652     \def\bbl@elt##1##2##3{%
1653       \ifnum\sfcode`##1=##3\relax
1654         \babel@savevariable{\sfcode`##1}%
1655         \sfcode`##1=##2\relax
```

---

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

```
1656        \fi}%
1657      \bbl@fs@chars
1658  \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.

```
1659 \bbl@trace{Short tags}
1660 \def\babeltags#1{%
1661   \edef\bbl@tempa{\zap@space#1 \@empty}%
1662   \def\bbl@tempb##1=##2\@@{%
1663     \edef\bbl@tempc{%
1664       \noexpand\newcommand
1665       \expandafter\noexpand\csname ##1\endcsname{%
1666         \noexpand\protect
1667         \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}%
1668       \noexpand\newcommand
1669       \expandafter\noexpand\csname text##1\endcsname{%
1670         \noexpand\foreignlanguage{##2}}}%
1671     \bbl@tempc}%
1672   \bbl@for\bbl@tempa\bbl@tempa{%
1673     \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.

```
1674 \bbl@trace{Hyphens}
1675 \@onlypreamble\babelhyphenation
1676 \AtEndOfPackage{%
1677   \newcommand\babelhyphenation[2][\@empty]{%
1678     \ifx\bbl@hyphenation@\relax
1679       \let\bbl@hyphenation@\@empty
1680     \fi
1681     \ifx\bbl@hyphlist\@empty\else
1682       \bbl@warning{%
1683         You must not intermingle \string\selectlanguage\space and\\%
1684         \string\babelhyphenation\space or some exceptions will not\\%
1685         be taken into account. Reported}%
1686     \fi
1687     \ifx\@empty#1%
1688       \protected@edef\bbl@hyphenation@{\bbl@hyphenation@\space#2}%
1689     \else
1690       \bbl@vforeach{#1}{%
1691         \def\bbl@tempa{##1}%
1692         \bbl@fixname\bbl@tempa
1693         \bbl@iflanguage\bbl@tempa{%
1694           \bbl@csarg\protected@edef{hyphenation@\bbl@tempa}{%
1695             \bbl@ifunset{bbl@hyphenation@\bbl@tempa}%
1696               {}%
1697               {\csname bbl@hyphenation@\bbl@tempa\endcsname\space}%
1698             #2}}}%
1699     \fi}}
```

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

```
1700 \def\bbl@allowhyphens{\ifvmode\else\nobreak\hskip\z@skip\fi}
```

---

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

```
1701 \def\bbl@t@one{T1}
1702 \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.

```
1703 \newcommand\babelnullhyphen{\char\hyphenchar\font}
1704 \def\babelhyphen{\active@prefix\babelhyphen\bbl@hyphen}
1705 \def\bbl@hyphen{%
1706   \@ifstar{\bbl@hyphen@i @}{\bbl@hyphen@i\@empty}}
1707 \def\bbl@hyphen@i#1#2{%
1708   \bbl@ifunset{bbl@hy@#1#2\@empty}%
1709     {\csname bbl@#1usehyphen\endcsname{\discretionary{#2}{}{#2}}}%
1710     {\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.

```
1711 \def\bbl@usehyphen#1{%
1712   \leavevmode
1713   \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
1714   \nobreak\hskip\z@skip}
1715 \def\bbl@@usehyphen#1{%
1716   \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}
```

The following macro inserts the hyphen char.

```
1717 \def\bbl@hyphenchar{%
1718   \ifnum\hyphenchar\font=\m@ne
1719     \babelnullhyphen
1720   \else
1721     \char\hyphenchar\font
1722   \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.

```
1723 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}{}}}
1724 \def\bbl@hy@@soft{\bbl@@usehyphen{\discretionary{\bbl@hyphenchar}{}{}}}
1725 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
1726 \def\bbl@hy@@hard{\bbl@@usehyphen\bbl@hyphenchar}
1727 \def\bbl@hy@nobreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}}
1728 \def\bbl@hy@@nobreak{\mbox{\bbl@hyphenchar}}
1729 \def\bbl@hy@repeat{%
1730   \bbl@usehyphen{%
1731     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
1732 \def\bbl@hy@@repeat{%
1733   \bbl@@usehyphen{%
1734     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
1735 \def\bbl@hy@empty{\hskip\z@skip}
1736 \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.

```
1737 \def\bbl@disc#1#2{\nobreak\discretionary{#2-}{}{#1}\bbl@allowhyphens}
```

7.10   Multiencoding strings

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

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

```
1738 \bbl@trace{Multiencoding strings}
1739 \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).

```
1740 \@ifpackagewith{babel}{nocase}%
1741   {\let\bbl@patchuclc\relax}%
1742   {\def\bbl@patchuclc{%
1743     \global\let\bbl@patchuclc\relax
1744     \g@addto@macro\@uclclist{\reserved@b{\reserved@b\bbl@uclc}}%
1745     \gdef\bbl@uclc##1{%
1746       \let\bbl@encoded\bbl@encoded@uclc
1747       \bbl@ifunset{\languagename @bbl@uclc}% and resumes it
1748         {##1}%
1749         {\let\bbl@tempa##1\relax % Used by LANG@bbl@uclc
1750          \csname\languagename @bbl@uclc\endcsname}%
1751       {\bbl@tolower\@empty}{\bbl@toupper\@empty}}%
1752     \gdef\bbl@tolower{\csname\languagename @bbl@lc\endcsname}%
1753     \gdef\bbl@toupper{\csname\languagename @bbl@uc\endcsname}}}
```

```
1754 ⟨⟨*More package options⟩⟩ ≡
1755 \DeclareOption{nocase}{}
1756 ⟨⟨/More package options⟩⟩
```

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

```
1757 ⟨⟨*More package options⟩⟩ ≡
1758 \let\bbl@opt@strings\@nnil % accept strings=value
1759 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
1760 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
1761 \def\BabelStringsDefault{generic}
1762 ⟨⟨/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.

```
1763 \@onlypreamble\StartBabelCommands
1764 \def\StartBabelCommands{%
1765   \begingroup
1766   \@tempcnta="7F
1767   \def\bbl@tempa{%
1768     \ifnum\@tempcnta>"FF\else
1769       \catcode\@tempcnta=11
1770       \advance\@tempcnta\@ne
1771       \expandafter\bbl@tempa
1772     \fi}%
1773   \bbl@tempa
1774   ⟨⟨Macros local to BabelCommands⟩⟩
1775   \def\bbl@provstring##1##2{%
1776     \providecommand##1{##2}%
1777     \bbl@toglobal##1}%
1778   \global\let\bbl@scafter\@empty
1779   \let\StartBabelCommands\bbl@startcmds
```

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

```
1803 \newcommand\bbl@startcmds@ii[1][\@empty]{%
1804    \let\SetString\@gobbletwo
1805    \let\bbl@stringdef\@gobbletwo
1806    \let\AfterBabelCommands\@gobble
1807    \ifx\@empty#1%
1808      \def\bbl@sc@label{generic}%
1809      \def\bbl@encstring##1##2{%
1810        \ProvideTextCommandDefault##1{##2}%
1811        \bbl@toglobal##1%
1812        \expandafter\bbl@toglobal\csname\string?\string##1\endcsname}%
1813      \let\bbl@sctest\in@true
1814    \else
1815      \let\bbl@sc@charset\space % <- zapped below
1816      \let\bbl@sc@fontenc\space % <-    "       "
1817      \def\bbl@tempa##1=##2\@nil{%
1818        \bbl@csarg\edef{sc@\zap@space##1 \@empty}{##2 }}%
1819      \bbl@vforeach{label=#1}{\bbl@tempa##1\@nil}%
1820      \def\bbl@tempa##1 ##2{% space -> comma
1821        ##1%
1822        \ifx\@empty##2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa##2\fi}%
1823      \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%
1824      \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%
1825      \edef\bbl@sc@charset{\expandafter\zap@space\bbl@sc@charset\@empty}%
1826      \def\bbl@encstring##1##2{%
1827        \bbl@foreach\bbl@sc@fontenc{%
1828          \bbl@ifunset{T@####1}%
1829            {}%
1830            {\ProvideTextCommand##1{####1}{##2}%
1831              \bbl@toglobal##1%
1832              \expandafter
```

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

```
1862 \def\bbl@forlang#1#2{%
1863   \bbl@for#1\bbl@L{%
1864     \bbl@xin@{,#1,}{,\BabelLanguages,}%
1865     \ifin@#2\relax\fi}}
1866 \def\bbl@scswitch{%
1867   \bbl@forlang\bbl@tempa{%
1868     \ifx\bbl@G\@empty\else
1869       \ifx\SetString\@gobbletwo\else
1870         \edef\bbl@GL{\bbl@G\bbl@tempa}%
1871         \bbl@xin@{,\bbl@GL,}{,\bbl@screset,}%
1872         \ifin@\else
1873           \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
1874           \xdef\bbl@screset{\bbl@screset,\bbl@GL}%
1875         \fi
1876       \fi
1877     \fi}}
1878 \AtEndOfPackage{%
1879   \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{}{#2}}}%
1880   \let\bbl@scswitch\relax}
1881 \@onlypreamble\EndBabelCommands
1882 \def\EndBabelCommands{%
1883   \bbl@usehooks{stopcommands}{}%
1884   \endgroup
1885   \endgroup
1886   \bbl@scafter}
```

```
1887 \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.

```
1888 \def\bbl@setstring#1#2{% eg, \prefacename{<string>}
1889   \bbl@forlang\bbl@tempa{%
1890     \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
1891     \bbl@ifunset{\bbl@LC}% eg, \germanchaptername
1892       {\bbl@exp{%
1893         \global\\\bbl@add\<\bbl@G\bbl@tempa>{\\\bbl@scset\\#1\<\bbl@LC>}}}%
1894       {}%
1895     \def\BabelString{#2}%
1896     \bbl@usehooks{stringprocess}{}%
1897     \expandafter\bbl@stringdef
1898       \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.

```
1899 \ifx\bbl@opt@strings\relax
1900   \def\bbl@scset#1#2{\def#1{\bbl@encoded#2}}
1901   \bbl@patchuclc
1902   \let\bbl@encoded\relax
1903   \def\bbl@encoded@uclc#1{%
1904     \@inmathwarn#1%
1905     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
1906       \expandafter\ifx\csname ?\string#1\endcsname\relax
1907         \TextSymbolUnavailable#1%
1908       \else
1909         \csname ?\string#1\endcsname
1910       \fi
1911     \else
1912       \csname\cf@encoding\string#1\endcsname
1913     \fi}
1914 \else
1915   \def\bbl@scset#1#2{\def#1{#2}}
1916 \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.

```
1917 ⟨⟨*Macros local to BabelCommands⟩⟩ ≡
1918 \def\SetStringLoop##1##2{%
1919   \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
1920   \count@\z@
1921   \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
1922     \advance\count@\@ne
1923     \toks@\expandafter{\bbl@tempa}%
1924     \bbl@exp{%
1925       \\\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
1926       \count@=\the\count@\relax}}%
1927 ⟨⟨/Macros local to BabelCommands⟩⟩
```

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

```
1928 \def\bbl@aftercmds#1{%
1929   \toks@\expandafter{\bbl@scafter#1}%
1930   \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.

```
1931 ⟨⟨∗Macros local to BabelCommands⟩⟩ ≡
1932   \newcommand\SetCase[3][]{%
1933     \bbl@patchuclc
1934     \bbl@forlang\bbl@tempa{%
1935       \bbl@carg\bbl@encstring{\bbl@tempa @bbl@uclc}{\bbl@tempa##1}%
1936       \bbl@carg\bbl@encstring{\bbl@tempa @bbl@uc}{##2}%
1937       \bbl@carg\bbl@encstring{\bbl@tempa @bbl@lc}{##3}}}%
1938 ⟨⟨/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.

```
1939 ⟨⟨∗Macros local to BabelCommands⟩⟩ ≡
1940   \newcommand\SetHyphenMap[1]{%
1941     \bbl@forlang\bbl@tempa{%
1942       \expandafter\bbl@stringdef
1943         \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
1944 ⟨⟨/Macros local to BabelCommands⟩⟩
```

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

```
1945 \newcommand\BabelLower[2]{% one to one.
1946   \ifnum\lccode#1=#2\else
1947     \babel@savevariable{\lccode#1}%
1948     \lccode#1=#2\relax
1949   \fi}
1950 \newcommand\BabelLowerMM[4]{% many-to-many
1951   \@tempcnta=#1\relax
1952   \@tempcntb=#4\relax
1953   \def\bbl@tempa{%
1954     \ifnum\@tempcnta>#2\else
1955       \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
1956       \advance\@tempcnta#3\relax
1957       \advance\@tempcntb#3\relax
1958       \expandafter\bbl@tempa
1959     \fi}%
1960   \bbl@tempa}
1961 \newcommand\BabelLowerMO[4]{% many-to-one
1962   \@tempcnta=#1\relax
1963   \def\bbl@tempa{%
1964     \ifnum\@tempcnta>#2\else
1965       \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
1966       \advance\@tempcnta#3
1967       \expandafter\bbl@tempa
1968     \fi}%
1969   \bbl@tempa}
```

The following package options control the behavior of hyphenation mapping.

```
1970 ⟨⟨∗More package options⟩⟩ ≡
1971 \DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\z@}
1972 \DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap\@ne}
1973 \DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\tw@}
1974 \DeclareOption{hyphenmap=other}{\chardef\bbl@opt@hyphenmap\thr@@}
1975 \DeclareOption{hyphenmap=other*}{\chardef\bbl@opt@hyphenmap4\relax}
1976 ⟨⟨/More package options⟩⟩
```

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

```
1977 \AtEndOfPackage{%
1978   \ifx\bbl@opt@hyphenmap\@undefined
1979     \bbl@xin@{,}{\bbl@language@opts}%
1980     \chardef\bbl@opt@hyphenmap\ifin@4\else\@ne\fi
1981   \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.

```
1982 \newcommand\setlocalecaption{%  TODO. Catch typos.
1983   \@ifstar\bbl@setcaption@s\bbl@setcaption@x}
1984 \def\bbl@setcaption@x#1#2#3{%  language caption-name string
1985   \bbl@trim@def\bbl@tempa{#2}%
1986   \bbl@xin@{.template}{\bbl@tempa}%
1987   \ifin@
1988     \bbl@ini@captions@template{#3}{#1}%
1989   \else
1990     \edef\bbl@tempd{%
1991       \expandafter\expandafter\expandafter
1992       \strip@prefix\expandafter\meaning\csname captions#1\endcsname}%
1993     \bbl@xin@
1994       {\expandafter\string\csname #2name\endcsname}%
1995       {\bbl@tempd}%
1996     \ifin@ % Renew caption
1997       \bbl@xin@{\string\bbl@scset}{\bbl@tempd}%
1998       \ifin@
1999         \bbl@exp{%
2000           \\\bbl@ifsamestring{\bbl@tempa}{\languagename}%
2001             {\\\bbl@scset\<#2name>\<#1#2name>}%
2002             {}}%
2003       \else % Old way converts to new way
2004         \bbl@ifunset{#1#2name}%
2005           {\bbl@exp{%
2006             \\\bbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}%
2007             \\\bbl@ifsamestring{\bbl@tempa}{\languagename}%
2008               {\def\<#2name>{\<#1#2name>}}%
2009               {}}}%
2010           {}%
2011       \fi
2012     \else
2013       \bbl@xin@{\string\bbl@scset}{\bbl@tempd}% New
2014       \ifin@ % New way
2015         \bbl@exp{%
2016           \\\bbl@add\<captions#1>{\\\bbl@scset\<#2name>\<#1#2name>}%
2017           \\\bbl@ifsamestring{\bbl@tempa}{\languagename}%
2018             {\\\bbl@scset\<#2name>\<#1#2name>}%
2019             {}}%
2020       \else  % Old way, but defined in the new way
2021         \bbl@exp{%
2022           \\\bbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}%
2023           \\\bbl@ifsamestring{\bbl@tempa}{\languagename}%
2024             {\def\<#2name>{\<#1#2name>}}%
2025             {}}%
2026       \fi%
2027     \fi
2028     \@namedef{#1#2name}{#3}%
2029     \toks@\expandafter{\bbl@captionslist}%
2030     \bbl@exp{\\\in@{\<#2name>}{\the\toks@}}%
2031     \ifin@\else
2032       \bbl@exp{\\\bbl@add\\\bbl@captionslist{\<#2name>}}%
2033       \bbl@toglobal\bbl@captionslist
2034     \fi
2035   \fi}
2036 % \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.

```
2037 \bbl@trace{Macros related to glyphs}
2038 \def\set@low@box#1{\setbox\tw@\hbox{,}\setbox\z@\hbox{#1}%
2039     \dimen\z@\ht\z@ \advance\dimen\z@ -\ht\tw@%
2040     \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.

```
2041 \def\save@sf@q#1{\leavevmode
2042   \begingroup
2043     \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
2044   \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.

```
2045 \ProvideTextCommand{\quotedblbase}{OT1}{%
2046   \save@sf@q{\set@low@box{\textquotedblright\/}%
2047     \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.

```
2048 \ProvideTextCommandDefault{\quotedblbase}{%
2049   \UseTextSymbol{OT1}{\quotedblbase}}
```

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

```
2050 \ProvideTextCommand{\quotesinglbase}{OT1}{%
2051   \save@sf@q{\set@low@box{\textquoteright\/}%
2052     \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.

```
2053 \ProvideTextCommandDefault{\quotesinglbase}{%
2054   \UseTextSymbol{OT1}{\quotesinglbase}}
```

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

```
2055 \ProvideTextCommand{\guillemetleft}{OT1}{%
2056   \ifmmode
2057     \ll
2058   \else
2059     \save@sf@q{\nobreak
2060       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2061   \fi}
2062 \ProvideTextCommand{\guillemetright}{OT1}{%
2063   \ifmmode
2064     \gg
2065   \else
2066     \save@sf@q{\nobreak
2067       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2068   \fi}
2069 \ProvideTextCommand{\guillemotleft}{OT1}{%
2070   \ifmmode
2071     \ll
2072   \else
```

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

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

```
2083 \ProvideTextCommandDefault{\guillemetleft}{%
2084   \UseTextSymbol{OT1}{\guillemetleft}}
2085 \ProvideTextCommandDefault{\guillemetright}{%
2086   \UseTextSymbol{OT1}{\guillemetright}}
2087 \ProvideTextCommandDefault{\guillemotleft}{%
2088   \UseTextSymbol{OT1}{\guillemotleft}}
2089 \ProvideTextCommandDefault{\guillemotright}{%
2090   \UseTextSymbol{OT1}{\guillemotright}}
```

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

```
2091 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2092   \ifmmode
2093     <%
2094   \else
2095     \save@sf@q{\nobreak
2096       \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
2097   \fi}
2098 \ProvideTextCommand{\guilsinglright}{OT1}{%
2099   \ifmmode
2100     >%
2101   \else
2102     \save@sf@q{\nobreak
2103       \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
2104   \fi}
```

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

```
2105 \ProvideTextCommandDefault{\guilsinglleft}{%
2106   \UseTextSymbol{OT1}{\guilsinglleft}}
2107 \ProvideTextCommandDefault{\guilsinglright}{%
2108   \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.

```
2109 \DeclareTextCommand{\ij}{OT1}{%
2110   i\kern-0.02em\bbl@allowhyphens j}
2111 \DeclareTextCommand{\IJ}{OT1}{%
2112   I\kern-0.02em\bbl@allowhyphens J}
2113 \DeclareTextCommand{\ij}{T1}{\char188}
2114 \DeclareTextCommand{\IJ}{T1}{\char156}
```

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

```
2115 \ProvideTextCommandDefault{\ij}{%
2116   \UseTextSymbol{OT1}{\ij}}
2117 \ProvideTextCommandDefault{\IJ}{%
2118   \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).

```
2119 \def\crrtic@{\hrule height0.1ex width0.3em}
2120 \def\crttic@{\hrule height0.1ex width0.33em}
2121 \def\ddj@{%
2122    \setbox0\hbox{d}\dimen@=\ht0
2123    \advance\dimen@1ex
2124    \dimen@.45\dimen@
2125    \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2126    \advance\dimen@ii.5ex
2127    \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2128 \def\DDJ@{%
2129    \setbox0\hbox{D}\dimen@=.55\ht0
2130    \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2131    \advance\dimen@ii.15ex %            correction for the dash position
2132    \advance\dimen@ii-.15\fontdimen7\font %     correction for cmtt font
2133    \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
2134    \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
2135 %
2136 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2137 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}
```

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

```
2138 \ProvideTextCommandDefault{\dj}{%
2139    \UseTextSymbol{OT1}{\dj}}
2140 \ProvideTextCommandDefault{\DJ}{%
2141    \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.

```
2142 \DeclareTextCommand{\SS}{OT1}{SS}
2143 \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
```
2144 \ProvideTextCommandDefault{\glq}{%
2145    \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}
```

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

```
2146 \ProvideTextCommand{\grq}{T1}{%
2147    \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}
2148 \ProvideTextCommand{\grq}{TU}{%
2149    \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
2150 \ProvideTextCommand{\grq}{OT1}{%
2151    \save@sf@q{\kern-.0125em
2152       \textormath{\textquoteleft}{\mbox{\textquoteleft}}%
2153       \kern.07em\relax}}
2154 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}\grq}
```

\glqq The 'german' double quotes.
\grqq
```
2155 \ProvideTextCommandDefault{\glqq}{%
2156    \textormath{\quotedblbase}{\mbox{\quotedblbase}}}
```

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

```
2157 \ProvideTextCommand{\grqq}{T1}{%
2158    \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2159 \ProvideTextCommand{\grqq}{TU}{%
2160    \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
```

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

\flq The 'french' single guillemets.
\frq
```
2166 \ProvideTextCommandDefault{\flq}{%
2167   \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
2168 \ProvideTextCommandDefault{\frq}{%
2169   \textormath{\guilsinglright}{\mbox{\guilsinglright}}}
```

\flqq The 'french' double guillemets.
\frqq
```
2170 \ProvideTextCommandDefault{\flqq}{%
2171   \textormath{\guillemetleft}{\mbox{\guillemetleft}}}
2172 \ProvideTextCommandDefault{\frqq}{%
2173   \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).

```
2174 \def\umlauthigh{%
2175   \def\bbl@umlauta##1{\leavevmode\bgroup%
2176     \accent\csname\f@encoding dqpos\endcsname
2177     ##1\bbl@allowhyphens\egroup}%
2178   \let\bbl@umlaute\bbl@umlauta}
2179 \def\umlautlow{%
2180   \def\bbl@umlauta{\protect\lower@umlaut}}
2181 \def\umlautelow{%
2182   \def\bbl@umlaute{\protect\lower@umlaut}}
2183 \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.

```
2184 \expandafter\ifx\csname U@D\endcsname\relax
2185   \csname newdimen\endcsname\U@D
2186 \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.

```
2187 \def\lower@umlaut#1{%
2188   \leavevmode\bgroup
2189     \U@D 1ex%
2190     {\setbox\z@\hbox{%
2191       \char\csname\f@encoding dqpos\endcsname}%
2192       \dimen@ -.45ex\advance\dimen@\ht\z@
2193       \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2194     \accent\csname\f@encoding dqpos\endcsname
2195     \fontdimen5\font\U@D #1%
2196   \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).

```
2197 \AtBeginDocument{%
2198   \DeclareTextCompositeCommand{\"}{OT1}{a}{\bbl@umlauta{a}}%
2199   \DeclareTextCompositeCommand{\"}{OT1}{e}{\bbl@umlaute{e}}%
2200   \DeclareTextCompositeCommand{\"}{OT1}{i}{\bbl@umlaute{\i}}%
2201   \DeclareTextCompositeCommand{\"}{OT1}{\i}{\bbl@umlaute{\i}}%
2202   \DeclareTextCompositeCommand{\"}{OT1}{o}{\bbl@umlauta{o}}%
2203   \DeclareTextCompositeCommand{\"}{OT1}{u}{\bbl@umlauta{u}}%
2204   \DeclareTextCompositeCommand{\"}{OT1}{A}{\bbl@umlauta{A}}%
2205   \DeclareTextCompositeCommand{\"}{OT1}{E}{\bbl@umlaute{E}}%
2206   \DeclareTextCompositeCommand{\"}{OT1}{I}{\bbl@umlaute{I}}%
2207   \DeclareTextCompositeCommand{\"}{OT1}{O}{\bbl@umlauta{O}}%
2208   \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.

```
2209 \ifx\l@english\@undefined
2210   \chardef\l@english\z@
2211 \fi
2212 % The following is used to cancel rules in ini files (see Amharic).
2213 \ifx\l@unhyphenated\@undefined
2214   \newlanguage\l@unhyphenated
2215 \fi
```

7.13 Layout

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

```
2216 \bbl@trace{Bidi layout}
2217 \providecommand\IfBabelLayout[3]{#3}%
2218 \newcommand\BabelPatchSection[1]{%
2219   \@ifundefined{#1}{}{%
2220     \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
2221     \@namedef{#1}{%
2222       \@ifstar{\bbl@presec@s{#1}}%
2223               {\@dblarg{\bbl@presec@x{#1}}}}}}
2224 \def\bbl@presec@x#1[#2]#3{%
2225   \bbl@exp{%
2226     \\\select@language@x{\bbl@main@language}%
2227     \\\bbl@cs{sspre@#1}%
2228     \\\bbl@cs{ss@#1}%
2229       [\\\foreignlanguage{\languagename}{\unexpanded{#2}}]%
2230       {\\\foreignlanguage{\languagename}{\unexpanded{#3}}}%
2231     \\\select@language@x{\languagename}}}
2232 \def\bbl@presec@s#1#2{%
2233   \bbl@exp{%
2234     \\\select@language@x{\bbl@main@language}%
2235     \\\bbl@cs{sspre@#1}%
2236     \\\bbl@cs{ss@#1}*%
2237       {\\\foreignlanguage{\languagename}{\unexpanded{#2}}}%
2238     \\\select@language@x{\languagename}}}
2239 \IfBabelLayout{sectioning}%
2240   {\BabelPatchSection{part}%
2241    \BabelPatchSection{chapter}%
2242    \BabelPatchSection{section}%
2243    \BabelPatchSection{subsection}%
2244    \BabelPatchSection{subsubsection}%
2245    \BabelPatchSection{paragraph}%
```

```
2246    \BabelPatchSection{subparagraph}%
2247    \def\babel@toc#1{%
2248      \select@language@x{\bbl@main@language}}}}{}
2249 \IfBabelLayout{captions}%
2250   {\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.

```
2251 \bbl@trace{Input engine specific macros}
2252 \ifcase\bbl@engine
2253   \input txtbabel.def
2254 \or
2255   \input luababel.def
2256 \or
2257   \input xebabel.def
2258 \fi
2259 \providecommand\babelfont{%
2260   \bbl@error
2261     {This macro is available only in LuaLaTeX and XeLaTeX.}%
2262     {Consider switching to these engines.}}
2263 \providecommand\babelprehyphenation{%
2264   \bbl@error
2265     {This macro is available only in LuaLaTeX.}%
2266     {Consider switching to that engine.}}
2267 \ifx\babelposthyphenation\@undefined
2268   \let\babelposthyphenation\babelprehyphenation
2269   \let\babelpatterns\babelprehyphenation
2270   \let\babelcharproperty\babelprehyphenation
2271 \fi
```

7.15 Creating and modifying languages

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

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

```
2297        \bbl@csarg\ifx{KVP@##1}\@nnil\else
2298          \bbl@error
2299            {Unknown key '##1' in \string\babelprovide}%
2300            {See the manual for valid keys}%
2301        \fi
2302        \bbl@csarg\def{KVP@##1}{##2}%
2303      \fi}%
2304  \chardef\bbl@howloaded=% 0:none; 1:ldf without ini; 2:ini
2305    \bbl@ifunset{date#2}\z@{\bbl@ifunset{bbl@llevel@#2}\@ne\tw@}%
2306  % == init ==
2307  \ifx\bbl@screset\@undefined
2308    \bbl@ldfinit
2309  \fi
2310  % == date (as option) ==
2311  % \ifx\bbl@KVP@date\@nnil\else
2312  % \fi
2313  % ==
2314  \let\bbl@lbkflag\relax % \@empty = do setup linebreak, only in 3 cases:
2315  \ifcase\bbl@howloaded
2316    \let\bbl@lbkflag\@empty % new
2317  \else
2318    \ifx\bbl@KVP@hyphenrules\@nnil\else
2319        \let\bbl@lbkflag\@empty
2320    \fi
2321    \ifx\bbl@KVP@import\@nnil\else
2322      \let\bbl@lbkflag\@empty
2323    \fi
2324  \fi
2325  % == import, captions ==
2326  \ifx\bbl@KVP@import\@nnil\else
2327    \bbl@exp{\\\bbl@ifblank{\bbl@KVP@import}}%
2328      {\ifx\bbl@initoload\relax
2329          \begingroup
2330            \def\BabelBeforeIni##1##2{\gdef\bbl@KVP@import{##1}\endinput}%
2331            \bbl@input@texini{#2}%
2332          \endgroup
2333        \else
2334          \xdef\bbl@KVP@import{\bbl@initoload}%
2335        \fi}%
2336      {}%
2337    \let\bbl@KVP@date\@empty
2338  \fi
2339  \let\bbl@KVP@captions@@\bbl@KVP@captions % TODO. A dirty hack
2340  \ifx\bbl@KVP@captions\@nnil
2341    \let\bbl@KVP@captions\bbl@KVP@import
2342  \fi
2343  % ==
2344  \ifx\bbl@KVP@transforms\@nnil\else
2345    \bbl@replace\bbl@KVP@transforms{ }{,}%
2346  \fi
2347  % == Load ini ==
2348  \ifcase\bbl@howloaded
2349    \bbl@provide@new{#2}%
2350  \else
2351    \bbl@ifblank{#1}%
2352      {}%  With \bbl@load@basic below
2353      {\bbl@provide@renew{#2}}%
2354  \fi
2355  % Post tasks
2356  % ----------
2357  % == subsequent calls after the first provide for a locale ==
2358  \ifx\bbl@inidata\@empty\else
2359    \bbl@extend@ini{#2}%
```

```
2360    \fi
2361    % == ensure captions ==
2362    \ifx\bbl@KVP@captions\@nnil\else
2363      \bbl@ifunset{bbl@extracaps@#2}%
2364        {\bbl@exp{\\\babelensure[exclude=\\\today]{#2}}}%
2365        {\bbl@exp{\\\babelensure[exclude=\\\today,
2366                  include=\[bbl@extracaps@#2]]{#2}}}%
2367      \bbl@ifunset{bbl@ensure@\languagename}%
2368        {\bbl@exp{%
2369          \\\DeclareRobustCommand\<bbl@ensure@\languagename>[1]{%
2370            \\\foreignlanguage{\languagename}%
2371            {####1}}}}%
2372        {}%
2373      \bbl@exp{%
2374        \\\bbl@toglobal\<bbl@ensure@\languagename>%
2375        \\\bbl@toglobal\<bbl@ensure@\languagename\space>}%
2376    \fi
2377    % ==
2378    % At this point all parameters are defined if 'import'. Now we
2379    % execute some code depending on them. But what about if nothing was
2380    % imported? We just set the basic parameters, but still loading the
2381    % whole ini file.
2382    \bbl@load@basic{#2}%
2383    % == script, language ==
2384    % Override the values from ini or defines them
2385    \ifx\bbl@KVP@script\@nnil\else
2386      \bbl@csarg\edef{sname@#2}{\bbl@KVP@script}%
2387    \fi
2388    \ifx\bbl@KVP@language\@nnil\else
2389      \bbl@csarg\edef{lname@#2}{\bbl@KVP@language}%
2390    \fi
2391    \ifcase\bbl@engine\or
2392      \bbl@ifunset{bbl@chrng@\languagename}{}%
2393        {\directlua{
2394          Babel.set_chranges_b('\bbl@cl{sbcp}', '\bbl@cl{chrng}') }}%
2395    \fi
2396     % == onchar ==
2397    \ifx\bbl@KVP@onchar\@nnil\else
2398      \bbl@luahyphenate
2399      \bbl@exp{%
2400        \\\AddToHook{env/document/before}{{\\\select@language{#2}{}}}}%
2401      \directlua{
2402        if Babel.locale_mapped == nil then
2403          Babel.locale_mapped = true
2404          Babel.linebreaking.add_before(Babel.locale_map, 1)
2405          Babel.loc_to_scr = {}
2406          Babel.chr_to_loc = Babel.chr_to_loc or {}
2407        end
2408        Babel.locale_props[\the\localeid].letters = false
2409      }%
2410      \bbl@xin@{ letters }{ \bbl@KVP@onchar\space}%
2411      \ifin@
2412        \directlua{
2413          Babel.locale_props[\the\localeid].letters = true
2414        }%
2415      \fi
2416      \bbl@xin@{ ids }{ \bbl@KVP@onchar\space}%
2417      \ifin@
2418        \ifx\bbl@starthyphens\@undefined % Needed if no explicit selection
2419          \AddBabelHook{babel-onchar}{beforestart}{{\bbl@starthyphens}}%
2420        \fi
2421        \bbl@exp{\\\bbl@add\\\bbl@starthyphens
2422          {\\\bbl@patterns@lua{\languagename}}}%
```

```
2423        % TODO - error/warning if no script
2424        \directlua{
2425          if Babel.script_blocks['\bbl@cl{sbcp}'] then
2426            Babel.loc_to_scr[\the\localeid] =
2427              Babel.script_blocks['\bbl@cl{sbcp}']
2428            Babel.locale_props[\the\localeid].lc = \the\localeid\space
2429            Babel.locale_props[\the\localeid].lg = \the\@nameuse{l@\languagename}\space
2430          end
2431        }%
2432      \fi
2433      \bbl@xin@{ fonts }{ \bbl@KVP@onchar\space}%
2434      \ifin@
2435        \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
2436        \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
2437        \directlua{
2438          if Babel.script_blocks['\bbl@cl{sbcp}'] then
2439            Babel.loc_to_scr[\the\localeid] =
2440              Babel.script_blocks['\bbl@cl{sbcp}']
2441          end}%
2442        \ifx\bbl@mapselect\@undefined  % TODO. almost the same as mapfont
2443          \AtBeginDocument{%
2444            \bbl@patchfont{{\bbl@mapselect}}%
2445            {\selectfont}}%
2446          \def\bbl@mapselect{%
2447            \let\bbl@mapselect\relax
2448            \edef\bbl@prefontid{\fontid\font}}%
2449          \def\bbl@mapdir##1{%
2450            {\def\languagename{##1}%
2451            \let\bbl@ifrestoring\@firstoftwo % To avoid font warning
2452            \bbl@switchfont
2453            \ifnum\fontid\font>\z@ % A hack, for the pgf nullfont hack
2454              \directlua{
2455                Babel.locale_props[\the\csname bbl@id@@##1\endcsname]%
2456                    ['/\bbl@prefontid'] = \fontid\font\space}%
2457            \fi}}%
2458        \fi
2459        \bbl@exp{\\\bbl@add\\\bbl@mapselect{\\\bbl@mapdir{\languagename}}}%
2460      \fi
2461      % TODO - catch non-valid values
2462    \fi
2463    % == mapfont ==
2464    % For bidi texts, to switch the font based on direction
2465    \ifx\bbl@KVP@mapfont\@nnil\else
2466      \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}{}%
2467        {\bbl@error{Option '\bbl@KVP@mapfont' unknown for\\%
2468                   mapfont. Use 'direction'.%
2469                  {See the manual for details.}}}%
2470      \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
2471      \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
2472      \ifx\bbl@mapselect\@undefined % TODO. See onchar.
2473        \AtBeginDocument{%
2474          \bbl@patchfont{{\bbl@mapselect}}%
2475          {\selectfont}}%
2476        \def\bbl@mapselect{%
2477          \let\bbl@mapselect\relax
2478          \edef\bbl@prefontid{\fontid\font}}%
2479        \def\bbl@mapdir##1{%
2480          {\def\languagename{##1}%
2481          \let\bbl@ifrestoring\@firstoftwo % avoid font warning
2482          \bbl@switchfont
2483          \directlua{Babel.fontmap
2484            [\the\csname bbl@wdir@##1\endcsname]%
2485            [\bbl@prefontid]=\fontid\font}}}%
```

116

```
2486     \fi
2487     \bbl@exp{\\\bbl@add\\\bbl@mapselect{\\\bbl@mapdir{\languagename}}}%
2488  \fi
2489  % == Line breaking: intraspace, intrapenalty ==
2490  % For CJK, East Asian, Southeast Asian, if interspace in ini
2491  \ifx\bbl@KVP@intraspace\@nnil\else % We can override the ini or set
2492    \bbl@csarg\edef{intsp@#2}{\bbl@KVP@intraspace}%
2493  \fi
2494  \bbl@provide@intraspace
2495  % == Line breaking: CJK quotes ==
2496  \ifcase\bbl@engine\or
2497    \bbl@xin@{/c}{/\bbl@cl{lnbrk}}%
2498    \ifin@
2499      \bbl@ifunset{bbl@quote@\languagename}{}%
2500        {\directlua{
2501          Babel.locale_props[\the\localeid].cjk_quotes = {}
2502          local cs = 'op'
2503          for c in string.utfvalues(%
2504              [[\csname bbl@quote@\languagename\endcsname]]) do
2505            if Babel.cjk_characters[c].c == 'qu' then
2506              Babel.locale_props[\the\localeid].cjk_quotes[c] = cs
2507            end
2508            cs = ( cs == 'op') and 'cl' or 'op'
2509          end
2510        }}%
2511    \fi
2512  \fi
2513  % == Line breaking: justification ==
2514  \ifx\bbl@KVP@justification\@nnil\else
2515    \let\bbl@KVP@linebreaking\bbl@KVP@justification
2516  \fi
2517  \ifx\bbl@KVP@linebreaking\@nnil\else
2518    \bbl@xin@{,\bbl@KVP@linebreaking,}%
2519      {,elongated,kashida,cjk,padding,unhyphenated,}%
2520    \ifin@
2521      \bbl@csarg\xdef
2522        {lnbrk@\languagename}{\expandafter\@car\bbl@KVP@linebreaking\@nil}%
2523    \fi
2524  \fi
2525  \bbl@xin@{/e}{/\bbl@cl{lnbrk}}%
2526  \ifin@\else\bbl@xin@{/k}{/\bbl@cl{lnbrk}}\fi
2527  \ifin@\bbl@arabicjust\fi
2528  \bbl@xin@{/p}{/\bbl@cl{lnbrk}}%
2529  \ifin@\AtBeginDocument{\@nameuse{bbl@tibetanjust}}\fi
2530  % == Line breaking: hyphenate.other.(locale|script) ==
2531  \ifx\bbl@lbkflag\@empty
2532    \bbl@ifunset{bbl@hyotl@\languagename}{}%
2533      {\bbl@csarg\bbl@replace{hyotl@\languagename}{ }{,}%
2534       \bbl@startcommands*{\languagename}{}%
2535         \bbl@csarg\bbl@foreach{hyotl@\languagename}{%
2536           \ifcase\bbl@engine
2537             \ifnum##1<257
2538               \SetHyphenMap{\BabelLower{##1}{##1}}%
2539             \fi
2540           \else
2541             \SetHyphenMap{\BabelLower{##1}{##1}}%
2542           \fi}%
2543       \bbl@endcommands}%
2544    \bbl@ifunset{bbl@hyots@\languagename}{}%
2545      {\bbl@csarg\bbl@replace{hyots@\languagename}{ }{,}%
2546       \bbl@csarg\bbl@foreach{hyots@\languagename}{%
2547         \ifcase\bbl@engine
2548           \ifnum##1<257
```

```
2549            \global\lccode##1=##1\relax
2550          \fi
2551        \else
2552          \global\lccode##1=##1\relax
2553        \fi}}%
2554  \fi
2555  % == Counters: maparabic ==
2556  % Native digits, if provided in ini (TeX level, xe and lua)
2557  \ifcase\bbl@engine\else
2558    \bbl@ifunset{bbl@dgnat@\languagename}{}%
2559      {\expandafter\ifx\csname bbl@dgnat@\languagename\endcsname\@empty\else
2560        \expandafter\expandafter\expandafter
2561        \bbl@setdigits\csname bbl@dgnat@\languagename\endcsname
2562        \ifx\bbl@KVP@maparabic\@nnil\else
2563          \ifx\bbl@latinarabic\@undefined
2564            \expandafter\let\expandafter\@arabic
2565              \csname bbl@counter@\languagename\endcsname
2566          \else    % ie, if layout=counters, which redefines \@arabic
2567            \expandafter\let\expandafter\bbl@latinarabic
2568              \csname bbl@counter@\languagename\endcsname
2569          \fi
2570        \fi
2571      \fi}%
2572  \fi
2573  % == Counters: mapdigits ==
2574  % > luababel.def
2575  % == Counters: alph, Alph ==
2576  \ifx\bbl@KVP@alph\@nnil\else
2577    \bbl@exp{%
2578      \\\bbl@add\<bbl@preextras@\languagename>{%
2579        \\\babel@save\\\@alph
2580        \let\\\@alph\<bbl@cntr@\bbl@KVP@alph @\languagename>}}%
2581  \fi
2582  \ifx\bbl@KVP@Alph\@nnil\else
2583    \bbl@exp{%
2584      \\\bbl@add\<bbl@preextras@\languagename>{%
2585        \\\babel@save\\\@Alph
2586        \let\\\@Alph\<bbl@cntr@\bbl@KVP@Alph @\languagename>}}%
2587  \fi
2588  % == Calendars ==
2589  \ifx\bbl@KVP@calendar\@nnil
2590    \edef\bbl@KVP@calendar{\bbl@cl{calpr}}%
2591  \fi
2592  \def\bbl@tempe##1 ##2\@@{% Get first calendar
2593    \def\bbl@tempa{##1}}%
2594    \bbl@exp{\\\bbl@tempe\bbl@KVP@calendar\space\\\@@}%
2595  \def\bbl@tempe##1.##2.##3\@@{%
2596    \def\bbl@tempc{##1}%
2597    \def\bbl@tempb{##2}}%
2598  \expandafter\bbl@tempe\bbl@tempa..\@@
2599  \bbl@csarg\edef{calpr@\languagename}{%
2600    \ifx\bbl@tempc\@empty\else
2601      calendar=\bbl@tempc
2602    \fi
2603    \ifx\bbl@tempb\@empty\else
2604      ,variant=\bbl@tempb
2605    \fi}%
2606  % == engine specific extensions ==
2607  % Defined in XXXbabel.def
2608  \bbl@provide@extra{#2}%
2609  % == require.babel in ini ==
2610  % To load or reaload the babel-*.tex, if require.babel in ini
2611  \ifx\bbl@beforestart\relax\else  % But not in doc aux or body
```

118

```
2612    \bbl@ifunset{bbl@rqtex@\languagename}{}%
2613      {\expandafter\ifx\csname bbl@rqtex@\languagename\endcsname\@empty\else
2614        \let\BabelBeforeIni\@gobbletwo
2615        \chardef\atcatcode=\catcode`\@
2616        \catcode`\@=11\relax
2617        \bbl@input@texini{\bbl@cs{rqtex@\languagename}}%
2618        \catcode`\@=\atcatcode
2619        \let\atcatcode\relax
2620        \global\bbl@csarg\let{rqtex@\languagename}\relax
2621      \fi}%
2622    \bbl@foreach\bbl@calendars{%
2623      \bbl@ifunset{bbl@ca@##1}{%
2624        \chardef\atcatcode=\catcode`\@
2625        \catcode`\@=11\relax
2626        \InputIfFileExists{babel-ca-##1.tex}{}{}%
2627        \catcode`\@=\atcatcode
2628        \let\atcatcode\relax}%
2629      {}}%
2630  \fi
2631  % == frenchspacing ==
2632  \ifcase\bbl@howloaded\in@true\else\in@false\fi
2633  \ifin@\else\bbl@xin@{typography/frenchspacing}{\bbl@key@list}\fi
2634  \ifin@
2635    \bbl@extras@wrap{\\\bbl@pre@fs}%
2636      {\bbl@pre@fs}%
2637      {\bbl@post@fs}%
2638  \fi
2639  % == transforms ==
2640  % > luababel.def
2641  % == main ==
2642  \ifx\bbl@KVP@main\@nnil  % Restore only if not 'main'
2643    \let\languagename\bbl@savelangname
2644    \chardef\localeid\bbl@savelocaleid\relax
2645  \fi}
```

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

```
2646  \def\bbl@provide@new#1{%
2647    \@namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
2648    \@namedef{extras#1}{}%
2649    \@namedef{noextras#1}{}%
2650    \bbl@startcommands*{#1}{captions}%
2651      \ifx\bbl@KVP@captions\@nnil %       and also if import, implicit
2652        \def\bbl@tempb##1{%              elt for \bbl@captionslist
2653          \ifx##1\@empty\else
2654            \bbl@exp{%
2655              \\\SetString\\##1{%
2656                \\\bbl@nocaption{\bbl@stripslash##1}{#1\bbl@stripslash##1}}}%
2657            \expandafter\bbl@tempb
2658          \fi}%
2659        \expandafter\bbl@tempb\bbl@captionslist\@empty
2660      \else
2661        \ifx\bbl@initoload\relax
2662          \bbl@read@ini{\bbl@KVP@captions}2%  % Here letters cat = 11
2663        \else
2664          \bbl@read@ini{\bbl@initoload}2%      % Same
2665        \fi
2666      \fi
2667    \StartBabelCommands*{#1}{date}%
2668      \ifx\bbl@KVP@date\@nnil
2669        \bbl@exp{%
2670          \\\SetString\\\today{\\\bbl@nocaption{today}{#1today}}}%
2671      \else
```

```
2672        \bbl@savetoday
2673        \bbl@savedate
2674      \fi
2675    \bbl@endcommands
2676    \bbl@load@basic{#1}%
2677    % == hyphenmins == (only if new)
2678    \bbl@exp{%
2679      \gdef\<#1hyphenmins>{%
2680        {\bbl@ifunset{bbl@lfthm@#1}{2}{\bbl@cs{lfthm@#1}}}%
2681        {\bbl@ifunset{bbl@rgthm@#1}{3}{\bbl@cs{rgthm@#1}}}}}%
2682    % == hyphenrules (also in renew) ==
2683    \bbl@provide@hyphens{#1}%
2684    \ifx\bbl@KVP@main\@nnil\else
2685      \expandafter\main@language\expandafter{#1}%
2686    \fi}
2687 %
2688 \def\bbl@provide@renew#1{%
2689    \ifx\bbl@KVP@captions\@nnil\else
2690      \StartBabelCommands*{#1}{captions}%
2691        \bbl@read@ini{\bbl@KVP@captions}2%    % Here all letters cat = 11
2692      \EndBabelCommands
2693    \fi
2694    \ifx\bbl@KVP@date\@nnil\else
2695      \StartBabelCommands*{#1}{date}%
2696        \bbl@savetoday
2697        \bbl@savedate
2698      \EndBabelCommands
2699    \fi
2700    % == hyphenrules (also in new) ==
2701    \ifx\bbl@lbkflag\@empty
2702      \bbl@provide@hyphens{#1}%
2703    \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.)

```
2704 \def\bbl@load@basic#1{%
2705    \ifcase\bbl@howloaded\or\or
2706      \ifcase\csname bbl@llevel@\languagename\endcsname
2707        \bbl@csarg\let{lname@\languagename}\relax
2708      \fi
2709    \fi
2710    \bbl@ifunset{bbl@lname@#1}%
2711      {\def\BabelBeforeIni##1##2{%
2712        \begingroup
2713          \let\bbl@ini@captions@aux\@gobbletwo
2714          \def\bbl@inidate ####1.####2.####3.####4\relax ####5####6{}%
2715          \bbl@read@ini{##1}1%
2716          \ifx\bbl@initoload\relax\endinput\fi
2717        \endgroup}%
2718      \begingroup        % boxed, to avoid extra spaces:
2719        \ifx\bbl@initoload\relax
2720          \bbl@input@texini{#1}%
2721        \else
2722          \setbox\z@\hbox{\BabelBeforeIni{\bbl@initoload}{}}%
2723        \fi
2724      \endgroup}%
2725      {}}
```

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.

```
2726 \def\bbl@provide@hyphens#1{%
2727    \@tempcnta\m@ne  % a flag
2728    \ifx\bbl@KVP@hyphenrules\@nnil\else
```

120

```
2729    \bbl@replace\bbl@KVP@hyphenrules{ }{,}%
2730    \bbl@foreach\bbl@KVP@hyphenrules{%
2731      \ifnum\@tempcnta=\m@ne    % if not yet found
2732        \bbl@ifsamestring{##1}{+}%
2733          {\bbl@carg\addlanguage{l@##1}}%
2734          {}%
2735        \bbl@ifunset{l@##1}% After a possible +
2736          {}%
2737          {\@tempcnta\@nameuse{l@##1}}%
2738      \fi}%
2739    \ifnum\@tempcnta=\m@ne
2740      \bbl@warning{%
2741        Requested 'hyphenrules=' for '\languagename' not found.\\%
2742        Using the default value. Reported}%
2743    \fi
2744  \fi
2745  \ifnum\@tempcnta=\m@ne           % if no opt or no language in opt found
2746    \ifx\bbl@KVP@captions@@\@nnil % TODO. Hackish. See above.
2747      \bbl@ifunset{bbl@hyphr@#1}{}% use value in ini, if exists
2748        {\bbl@exp{\\\bbl@ifblank{\bbl@cs{hyphr@#1}}}%
2749          {}%
2750          {\bbl@ifunset{l@\bbl@cl{hyphr}}%
2751            {}%                      if hyphenrules found:
2752            {\@tempcnta\@nameuse{l@\bbl@cl{hyphr}}}}}%
2753    \fi
2754  \fi
2755  \bbl@ifunset{l@#1}%
2756    {\ifnum\@tempcnta=\m@ne
2757       \bbl@carg\adddialect{l@#1}\language
2758     \else
2759       \bbl@carg\adddialect{l@#1}\@tempcnta
2760     \fi}%
2761    {\ifnum\@tempcnta=\m@ne\else
2762       \global\bbl@carg\chardef{l@#1}\@tempcnta
2763     \fi}}
```

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

```
2764 \def\bbl@input@texini#1{%
2765   \bbl@bsphack
2766     \bbl@exp{%
2767       \catcode`\\\%=14 \catcode`\\\\=0
2768       \catcode`\\\{=1  \catcode`\\\}=2
2769       \lowercase{\\\InputIfFileExists{babel-#1.tex}{}{}}%
2770       \catcode`\\\%=\the\catcode`\%\relax
2771       \catcode`\\\\=\the\catcode`\\\relax
2772       \catcode`\\\{=\the\catcode`\{\relax
2773       \catcode`\\\}=\the\catcode`\}\relax}%
2774   \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`.

```
2775 \def\bbl@iniline#1\bbl@iniline{%
2776   \@ifnextchar[\bbl@inisect{\@ifnextchar;\bbl@iniskip\bbl@inistore}#1\@@}% ]
2777 \def\bbl@inisect[#1]#2\@@{\def\bbl@section{#1}}
2778 \def\bbl@iniskip#1\@@{}%       if starts with ;
2779 \def\bbl@inistore#1=#2\@@{%      full (default)
2780   \bbl@trim@def\bbl@tempa{#1}%
2781   \bbl@trim\toks@{#2}%
2782   \bbl@xin@{;\bbl@section/\bbl@tempa;}{\bbl@key@list}%
2783   \ifin@\else
2784     \bbl@xin@{,identification/include.}%
2785             {,\bbl@section/\bbl@tempa}%
2786     \ifin@\edef\bbl@required@inis{\the\toks@}\fi
```

```
2787    \bbl@exp{%
2788      \\\g@addto@macro\\\bbl@inidata{%
2789        \\\bbl@elt{\bbl@section}{\bbl@tempa}{\the\toks@}}}%
2790    \fi}
2791 \def\bbl@inistore@min#1=#2\@@{%  minimal (maybe set in \bbl@read@ini)
2792    \bbl@trim@def\bbl@tempa{#1}%
2793    \bbl@trim\toks@{#2}%
2794    \bbl@xin@{.identification.}{.\bbl@section.}%
2795    \ifin@
2796      \bbl@exp{\\\g@addto@macro\\\bbl@inidata{%
2797        \\\bbl@elt{identification}{\bbl@tempa}{\the\toks@}}}%
2798    \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.

```
2799 \def\bbl@loop@ini{%
2800    \loop
2801      \if T\ifeof\bbl@readstream F\fi T\relax % Trick, because inside \loop
2802        \endlinechar\m@ne
2803        \read\bbl@readstream to \bbl@line
2804        \endlinechar`\^^M
2805        \ifx\bbl@line\@empty\else
2806          \expandafter\bbl@iniline\bbl@line\bbl@iniline
2807        \fi
2808    \repeat}
2809 \ifx\bbl@readstream\@undefined
2810    \csname newread\endcsname\bbl@readstream
2811 \fi
2812 \def\bbl@read@ini#1#2{%
2813    \global\let\bbl@extend@ini\@gobble
2814    \openin\bbl@readstream=babel-#1.ini
2815    \ifeof\bbl@readstream
2816      \bbl@error
2817        {There is no ini file for the requested language\\%
2818         (#1: \languagename). Perhaps you misspelled it or your\\%
2819         installation is not complete.}%
2820        {Fix the name or reinstall babel.}%
2821    \else
2822      % == Store ini data in \bbl@inidata ==
2823      \catcode`\[=12 \catcode`\]=12 \catcode`\==12 \catcode`\&=12
2824      \catcode`\;=12 \catcode`\|=12 \catcode`\%=14 \catcode`\-=12
2825      \bbl@info{Importing
2826                \ifcase#2font and identification \or basic \fi
2827                data for \languagename\\%
2828              from babel-#1.ini. Reported}%
2829      \ifnum#2=\z@
2830        \global\let\bbl@inidata\@empty
2831        \let\bbl@inistore\bbl@inistore@min    % Remember it's local
2832      \fi
2833      \def\bbl@section{identification}%
2834      \let\bbl@required@inis\@empty
2835      \bbl@exp{\\\bbl@inistore tag.ini=#1\\\@@}%
2836      \bbl@inistore load.level=#2\@@
2837      \bbl@loop@ini
2838      \ifx\bbl@required@inis\@empty\else
2839        \bbl@replace\bbl@required@inis{ }{,}%
2840        \bbl@foreach\bbl@required@inis{%
2841          \openin\bbl@readstream=##1.ini
2842          \bbl@loop@ini}%
```

```
2843        \fi
2844    % == Process stored data ==
2845    \bbl@csarg\xdef{lini@\languagename}{#1}%
2846    \bbl@read@ini@aux
2847    % == 'Export' data ==
2848    \bbl@ini@exports{#2}%
2849    \global\bbl@csarg\let{inidata@\languagename}\bbl@inidata
2850    \global\let\bbl@inidata\@empty
2851    \bbl@exp{\\\bbl@add@list\\\bbl@ini@loaded{\languagename}}%
2852    \bbl@toglobal\bbl@ini@loaded
2853  \fi}
2854 \def\bbl@read@ini@aux{%
2855  \let\bbl@savestrings\@empty
2856  \let\bbl@savetoday\@empty
2857  \let\bbl@savedate\@empty
2858  \def\bbl@elt##1##2##3{%
2859    \def\bbl@section{##1}%
2860    \in@{=date.}{=##1}% Find a better place
2861    \ifin@
2862      \bbl@ifunset{bbl@inikv@##1}%
2863        {\bbl@ini@calendar{##1}}%
2864        {}%
2865    \fi
2866    \in@{=identification/extension.}{=##1/##2}%
2867    \ifin@
2868      \bbl@ini@extension{##2}%
2869    \fi
2870    \bbl@ifunset{bbl@inikv@##1}{}%
2871      {\csname bbl@inikv@##1\endcsname{##2}{##3}}}%
2872  \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.

```
2873 \def\bbl@extend@ini@aux#1{%
2874  \bbl@startcommands*{#1}{captions}%
2875    % Activate captions/... and modify exports
2876    \bbl@csarg\def{inikv@captions.licr}##1##2{%
2877      \setlocalecaption{#1}{##1}{##2}}%
2878    \def\bbl@inikv@captions##1##2{%
2879      \bbl@ini@captions@aux{##1}{##2}}%
2880    \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2881    \def\bbl@exportkey##1##2##3{%
2882      \bbl@ifunset{bbl@@kv@##2}{}%
2883        {\expandafter\ifx\csname bbl@@kv@##2\endcsname\@empty\else
2884          \bbl@exp{\global\let\<bbl@##1@\languagename>\<bbl@@kv@##2>}%
2885        \fi}}%
2886    % As with \bbl@read@ini, but with some changes
2887    \bbl@read@ini@aux
2888    \bbl@ini@exports\tw@
2889    % Update inidata@lang by pretending the ini is read.
2890    \def\bbl@elt##1##2##3{%
2891      \def\bbl@section{##1}%
2892      \bbl@iniline##2=##3\bbl@iniline}%
2893    \csname bbl@inidata@#1\endcsname
2894    \global\bbl@csarg\let{inidata@#1}\bbl@inidata
2895  \StartBabelCommands*{#1}{date}% And from the import stuff
2896    \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2897    \bbl@savetoday
2898    \bbl@savedate
2899  \bbl@endcommands}
```

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

```
2900 \def\bbl@ini@calendar#1{%
2901  \lowercase{\def\bbl@tempa{=#1=}}%
```

```
2902 \bbl@replace\bbl@tempa{=date.gregorian}{}%
2903 \bbl@replace\bbl@tempa{=date.}{}%
2904 \in@{.licr=}{#1=}%
2905 \ifin@
2906   \ifcase\bbl@engine
2907     \bbl@replace\bbl@tempa{.licr=}{}%
2908   \else
2909     \let\bbl@tempa\relax
2910   \fi
2911 \fi
2912 \ifx\bbl@tempa\relax\else
2913   \bbl@replace\bbl@tempa{=}{}%
2914   \ifx\bbl@tempa\@empty\else
2915     \xdef\bbl@calendars{\bbl@calendars,\bbl@tempa}%
2916   \fi
2917   \bbl@exp{%
2918     \def\<bbl@inikv@#1>####1####2{%
2919       \\\bbl@inidate####1...\relax{####2}{\bbl@tempa}}}%
2920 \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).

```
2921 \def\bbl@renewinikey#1/#2\@@#3{%
2922   \edef\bbl@tempa{\zap@space #1 \@empty}%   section
2923   \edef\bbl@tempb{\zap@space #2 \@empty}%   key
2924   \bbl@trim\toks@{#3}%                      value
2925   \bbl@exp{%
2926     \edef\\\bbl@key@list{\bbl@key@list \bbl@tempa/\bbl@tempb;}%
2927     \\\g@addto@macro\\\bbl@inidata{%
2928       \\\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.

```
2929 \def\bbl@exportkey#1#2#3{%
2930   \bbl@ifunset{bbl@@kv@#2}%
2931     {\bbl@csarg\gdef{#1@\languagename}{#3}}%
2932     {\expandafter\ifx\csname bbl@@kv@#2\endcsname\@empty
2933       \bbl@csarg\gdef{#1@\languagename}{#3}%
2934     \else
2935       \bbl@exp{\global\let\<bbl@#1@\languagename>\<bbl@@kv@#2>}%
2936     \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.

```
2937 \def\bbl@iniwarning#1{%
2938   \bbl@ifunset{bbl@@kv@identification.warning#1}{}%
2939     {\bbl@warning{%
2940       From babel-\bbl@cs{lini@\languagename}.ini:\\%
2941       \bbl@cs{@kv@identification.warning#1}\\%
2942       Reported }}}
2943 %
2944 \let\bbl@release@transforms\@empty
```

BCP 47 extensions are separated by a single letter (eg, latin-x-medieval. The following macro handles this special case to create correctly the correspondig info.

```
2945 \def\bbl@ini@extension#1{%
2946   \def\bbl@tempa{#1}%
2947   \bbl@replace\bbl@tempa{extension.}{}%
2948   \bbl@replace\bbl@tempa{.tag.bcp47}{}%
2949   \bbl@ifunset{bbl@info@#1}%
2950     {\bbl@csarg\xdef{info@#1}{ext/\bbl@tempa}%
```

```
2951      \bbl@exp{%
2952        \\\g@addto@macro\\\bbl@moreinfo{%
2953          \\\bbl@exportkey{ext/\bbl@tempa}{identification.#1}{}}}}%
2954    {}}
2955 \let\bbl@moreinfo\@empty
2956 %
2957 \def\bbl@ini@exports#1{%
2958   % Identification always exported
2959   \bbl@iniwarning{}%
2960   \ifcase\bbl@engine
2961     \bbl@iniwarning{.pdflatex}%
2962   \or
2963     \bbl@iniwarning{.lualatex}%
2964   \or
2965     \bbl@iniwarning{.xelatex}%
2966   \fi%
2967   \bbl@exportkey{llevel}{identification.load.level}{}%
2968   \bbl@exportkey{elname}{identification.name.english}{}%
2969   \bbl@exp{\\\bbl@exportkey{lname}{identification.name.opentype}%
2970     {\csname bbl@elname@\languagename\endcsname}}%
2971   \bbl@exportkey{tbcp}{identification.tag.bcp47}{}%
2972   \bbl@exportkey{lbcp}{identification.language.tag.bcp47}{}%
2973   \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
2974   \bbl@exportkey{esname}{identification.script.name}{}%
2975   \bbl@exp{\\\bbl@exportkey{sname}{identification.script.name.opentype}%
2976     {\csname bbl@esname@\languagename\endcsname}}%
2977   \bbl@exportkey{sbcp}{identification.script.tag.bcp47}{}%
2978   \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
2979   \bbl@exportkey{rbcp}{identification.region.tag.bcp47}{}%
2980   \bbl@exportkey{vbcp}{identification.variant.tag.bcp47}{}%
2981   \bbl@moreinfo
2982   % Also maps bcp47 -> languagename
2983   \ifbbl@bcptoname
2984     \bbl@csarg\xdef{bcp@map@\bbl@cl{tbcp}}{\languagename}%
2985   \fi
2986   % Conditional
2987   \ifnum#1>\z@          % 0 = only info, 1, 2 = basic, (re)new
2988     \bbl@exportkey{calpr}{date.calendar.preferred}{}%
2989     \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
2990     \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
2991     \bbl@exportkey{lfthm}{typography.lefthyphenmin}{2}%
2992     \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
2993     \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
2994     \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
2995     \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
2996     \bbl@exportkey{intsp}{typography.intraspace}{}%
2997     \bbl@exportkey{frspc}{typography.frenchspacing}{u}%
2998     \bbl@exportkey{chrng}{characters.ranges}{}%
2999     \bbl@exportkey{quote}{characters.delimiters.quotes}{}%
3000     \bbl@exportkey{dgnat}{numbers.digits.native}{}%
3001     \ifnum#1=\tw@          % only (re)new
3002       \bbl@exportkey{rqtex}{identification.require.babel}{}%
3003       \bbl@toglobal\bbl@savetoday
3004       \bbl@toglobal\bbl@savedate
3005       \bbl@savestrings
3006     \fi
3007   \fi}
```

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

```
3008 \def\bbl@inikv#1#2{%       key=value
3009   \toks@{#2}%               This hides #'s from ini values
3010   \bbl@csarg\edef{@kv@\bbl@section.#1}{\the\toks@}}
```

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

```
3011 \let\bbl@inikv@identification\bbl@inikv
3012 \let\bbl@inikv@date\bbl@inikv
3013 \let\bbl@inikv@typography\bbl@inikv
3014 \let\bbl@inikv@characters\bbl@inikv
3015 \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'.

```
3016 \def\bbl@inikv@counters#1#2{%
3017   \bbl@ifsamestring{#1}{digits}%
3018     {\bbl@error{The counter name 'digits' is reserved for mapping\\%
3019                 decimal digits}%
3020                {Use another name.}}%
3021     {}%
3022   \def\bbl@tempc{#1}%
3023   \bbl@trim@def{\bbl@tempb*}{#2}%
3024   \in@{.1$}{#1$}%
3025   \ifin@
3026     \bbl@replace\bbl@tempc{.1}{}%
3027     \bbl@csarg\protected@xdef{cntr@\bbl@tempc @\languagename}{%
3028       \noexpand\bbl@alphnumeral{\bbl@tempc}}%
3029   \fi
3030   \in@{.F.}{#1}%
3031   \ifin@\else\in@{.S.}{#1}\fi
3032   \ifin@
3033     \bbl@csarg\protected@xdef{cntr@#1@\languagename}{\bbl@tempb*}%
3034   \else
3035     \toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa
3036     \expandafter\bbl@buildifcase\bbl@tempb* \\ % Space after \\
3037     \bbl@csarg{\global\expandafter\let}{cntr@#1@\languagename}\bbl@tempa
3038   \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.

```
3039 \ifcase\bbl@engine
3040   \bbl@csarg\def{inikv@captions.licr}#1#2{%
3041     \bbl@ini@captions@aux{#1}{#2}}
3042 \else
3043   \def\bbl@inikv@captions#1#2{%
3044     \bbl@ini@captions@aux{#1}{#2}}
3045 \fi
```

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

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

```
3065          {\\\@nameuse{bbl@\bbl@tempa fmt@\\\languagename}}}}%
3066   \fi}
3067 \def\bbl@ini@captions@aux#1#2{%
3068   \bbl@trim@def\bbl@tempa{#1}%
3069   \bbl@xin@{.template}{\bbl@tempa}%
3070   \ifin@
3071     \bbl@ini@captions@template{#2}\languagename
3072   \else
3073     \bbl@ifblank{#2}%
3074       {\bbl@exp{%
3075         \toks@{\\\bbl@nocaption{\bbl@tempa}{\languagename\bbl@tempa name}}}}%
3076       {\bbl@trim\toks@{#2}}%
3077     \bbl@exp{%
3078       \\\bbl@add\\\bbl@savestrings{%
3079         \\\SetString\<\bbl@tempa name>{\the\toks@}}}%
3080     \toks@\expandafter{\bbl@captionslist}%
3081     \bbl@exp{\\\in@{\<\bbl@tempa name>}{\the\toks@}}%
3082     \ifin@\else
3083       \bbl@exp{%
3084         \\\bbl@add\<bbl@extracaps@\languagename>{\<\bbl@tempa name>}%
3085         \\\bbl@toglobal\<bbl@extracaps@\languagename>}%
3086     \fi
3087   \fi}
```

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

```
3088 \def\bbl@list@the{%
3089   part,chapter,section,subsection,subsubsection,paragraph,%
3090   subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,%
3091   table,page,footnote,mpfootnote,mpfn}
3092 \def\bbl@map@cnt#1{%  #1:roman,etc, // #2:enumi,etc
3093   \bbl@ifunset{bbl@map@#1@\languagename}%
3094     {\@nameuse{#1}}%
3095     {\@nameuse{bbl@map@#1@\languagename}}}
3096 \def\bbl@inikv@labels#1#2{%
3097   \in@{.map}{#1}%
3098   \ifin@
3099     \ifx\bbl@KVP@labels\@nnil\else
3100       \bbl@xin@{ map }{ \bbl@KVP@labels\space}%
3101       \ifin@
3102         \def\bbl@tempc{#1}%
3103         \bbl@replace\bbl@tempc{.map}{}%
3104         \in@{,#2,}{,arabic,roman,Roman,alph,Alph,fnsymbol,}%
3105         \bbl@exp{%
3106           \gdef\<bbl@map@\bbl@tempc @\languagename>%
3107             {\ifin@\<#2>\else\\\localcounter{#2}\fi}}%
3108         \bbl@foreach\bbl@list@the{%
3109           \bbl@ifunset{the##1}{}%
3110             {\bbl@exp{\let\\\bbl@tempd\<the##1>}%
3111              \bbl@exp{%
3112                \\\bbl@sreplace\<the##1>%
3113                  {\<\bbl@tempc>{##1}}{\\\bbl@map@cnt{\bbl@tempc}{##1}}%
3114                \\\bbl@sreplace\<the##1>%
3115                  {\<\@empty @\bbl@tempc>\<c@##1>}{\\\bbl@map@cnt{\bbl@tempc}{##1}}}%
3116             \expandafter\ifx\csname the##1\endcsname\bbl@tempd\else
3117               \toks@\expandafter\expandafter\expandafter{%
3118                 \csname the##1\endcsname}%
3119               \expandafter\xdef\csname the##1\endcsname{{\the\toks@}}%
3120             \fi}}%
3121       \fi
3122     \fi
3123   %
3124   \else
3125     %
```

127

```
3126    % The following code is still under study. You can test it and make
3127    % suggestions. Eg, enumerate.2 = ([enumi]).([enumii]). It's
3128    % language dependent.
3129    \in@{enumerate.}{#1}%
3130    \ifin@
3131      \def\bbl@tempa{#1}%
3132      \bbl@replace\bbl@tempa{enumerate.}{}%
3133      \def\bbl@toreplace{#2}%
3134      \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
3135      \bbl@replace\bbl@toreplace{[}{\csname the}%
3136      \bbl@replace\bbl@toreplace{]}{\endcsname{}}%
3137      \toks@\expandafter{\bbl@toreplace}%
3138      % TODO. Execute only once:
3139      \bbl@exp{%
3140        \\\bbl@add\<extras\languagename>{%
3141          \\\babel@save\<labelenum\romannumeral\bbl@tempa>%
3142          \def\<labelenum\romannumeral\bbl@tempa>{\the\toks@}}%
3143        \\\bbl@toglobal\<extras\languagename>}%
3144    \fi
3145  \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.

```
3146 \def\bbl@chaptype{chapter}
3147 \ifx\@makechapterhead\@undefined
3148   \let\bbl@patchchapter\relax
3149 \else\ifx\thechapter\@undefined
3150   \let\bbl@patchchapter\relax
3151 \else\ifx\ps@headings\@undefined
3152   \let\bbl@patchchapter\relax
3153 \else
3154   \def\bbl@patchchapter{%
3155     \global\let\bbl@patchchapter\relax
3156     \gdef\bbl@chfmt{%
3157       \bbl@ifunset{bbl@\bbl@chaptype fmt@\languagename}%
3158         {\@chapapp\space\thechapter}
3159         {\@nameuse{bbl@\bbl@chaptype fmt@\languagename}}}
3160     \bbl@add\appendix{\def\bbl@chaptype{appendix}}% Not harmful, I hope
3161     \bbl@sreplace\ps@headings{\@chapapp\ \thechapter}{\bbl@chfmt}%
3162     \bbl@sreplace\chaptermark{\@chapapp\ \thechapter}{\bbl@chfmt}%
3163     \bbl@sreplace\@makechapterhead{\@chapapp\space\thechapter}{\bbl@chfmt}%
3164     \bbl@toglobal\appendix
3165     \bbl@toglobal\ps@headings
3166     \bbl@toglobal\chaptermark
3167     \bbl@toglobal\@makechapterhead}
3168   \let\bbl@patchappendix\bbl@patchchapter
3169 \fi\fi\fi
3170 \ifx\@part\@undefined
3171   \let\bbl@patchpart\relax
3172 \else
3173   \def\bbl@patchpart{%
3174     \global\let\bbl@patchpart\relax
3175     \gdef\bbl@partformat{%
3176       \bbl@ifunset{bbl@partfmt@\languagename}%
3177         {\partname\nobreakspace\thepart}
3178         {\@nameuse{bbl@partfmt@\languagename}}}
3179     \bbl@sreplace\@part{\partname\nobreakspace\thepart}{\bbl@partformat}%
3180     \bbl@toglobal\@part}
3181 \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

```
3182 \let\bbl@calendar\@empty
3183 \DeclareRobustCommand\localedate[1][]{\bbl@localedate{#1}}
3184 \def\bbl@localedate#1#2#3#4{%
3185   \begingroup
3186     \edef\bbl@they{#2}%
3187     \edef\bbl@them{#3}%
3188     \edef\bbl@thed{#4}%
3189     \edef\bbl@tempe{%
3190       \bbl@ifunset{bbl@calpr@\languagename}{}{\bbl@cl{calpr}},%
3191       #1}%
3192     \bbl@replace\bbl@tempe{ }{}%
3193     \bbl@replace\bbl@tempe{CONVERT}{convert=}% Hackish
3194     \bbl@replace\bbl@tempe{convert}{convert=}%
3195     \let\bbl@ld@calendar\@empty
3196     \let\bbl@ld@variant\@empty
3197     \let\bbl@ld@convert\relax
3198     \def\bbl@tempb##1=##2\@@{\@namedef{bbl@ld@##1}{##2}}%
3199     \bbl@foreach\bbl@tempe{\bbl@tempb##1\@@}%
3200     \bbl@replace\bbl@ld@calendar{gregorian}{}%
3201     \ifx\bbl@ld@calendar\@empty\else
3202       \ifx\bbl@ld@convert\relax\else
3203         \babelcalendar[\bbl@they-\bbl@them-\bbl@thed]%
3204           {\bbl@ld@calendar}\bbl@they\bbl@them\bbl@thed
3205       \fi
3206     \fi
3207     \@nameuse{bbl@precalendar}% Remove, eg, +, -civil (-ca-islamic)
3208     \edef\bbl@calendar{% Used in \month..., too
3209       \bbl@ld@calendar
3210       \ifx\bbl@ld@variant\@empty\else
3211         .\bbl@ld@variant
3212       \fi}%
3213     \bbl@cased
3214       {\@nameuse{bbl@date@\languagename @\bbl@calendar}%
3215         \bbl@they\bbl@them\bbl@thed}%
3216   \endgroup}
3217 % eg: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
3218 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
3219   \bbl@trim@def\bbl@tempa{#1.#2}%
3220   \bbl@ifsamestring{\bbl@tempa}{months.wide}%      to savedate
3221     {\bbl@trim@def\bbl@tempa{#3}%
3222     \bbl@trim\toks@{#5}%
3223     \@temptokena\expandafter{\bbl@savedate}%
3224     \bbl@exp{%   Reverse order - in ini last wins
3225       \def\\\bbl@savedate{%
3226         \\\SetString\<month\romannumeral\bbl@tempa#6name>{\the\toks@}%
3227         \the\@temptokena}}}%
3228   {\bbl@ifsamestring{\bbl@tempa}{date.long}%     defined now
3229     {\lowercase{\def\bbl@tempb{#6}}%
3230     \bbl@trim@def\bbl@toreplace{#5}%
3231     \bbl@TG@@date
3232     \global\bbl@csarg\let{date@\languagename @\bbl@tempb}\bbl@toreplace
3233     \ifx\bbl@savetoday\@empty
3234       \bbl@exp{% TODO. Move to a better place.
3235         \\\AfterBabelCommands{%
3236           \def\<\languagename date>{\\\protect\<\languagename date >}%
3237           \\\newcommand\<\languagename date >[4][]{%
3238             \\\bbl@usedategrouptrue
3239             \<bbl@ensure@\languagename>{%
3240               \\\localedate[####1]{####2}{####3}{####4}}}}%
3241         \def\\\bbl@savetoday{%
3242           \\\SetString\\\today{%
3243             \<\languagename date>[convert]%
3244               {\\\the\year}{\\\the\month}{\\\the\day}}}}%
```

```
3245        \fi}%
3246        {}}}
```

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).

```
3247 \let\bbl@calendar\@empty
3248 \newcommand\babelcalendar[2][\the\year-\the\month-\the\day]{%
3249   \@nameuse{bbl@ca@#2}#1\@@}
3250 \newcommand\BabelDateSpace{\nobreakspace}
3251 \newcommand\BabelDateDot{.\@}  % TODO. \let instead of repeating
3252 \newcommand\BabelDated[1]{{\number#1}}
3253 \newcommand\BabelDatedd[1]{{\ifnum#1<10 0\fi\number#1}}
3254 \newcommand\BabelDateM[1]{{\number#1}}
3255 \newcommand\BabelDateMM[1]{{\ifnum#1<10 0\fi\number#1}}
3256 \newcommand\BabelDateMMMM[1]{{%
3257   \csname month\romannumeral#1\bbl@calendar name\endcsname}}%
3258 \newcommand\BabelDatey[1]{{\number#1}}%
3259 \newcommand\BabelDateyy[1]{{%
3260   \ifnum#1<10 0\number#1 %
3261   \else\ifnum#1<100 \number#1 %
3262   \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3263   \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
3264   \else
3265     \bbl@error
3266       {Currently two-digit years are restricted to the\\
3267        range 0-9999.}%
3268       {There is little you can do. Sorry.}%
3269   \fi\fi\fi\fi}}
3270 \newcommand\BabelDateyyyy[1]{{\number#1}} % TODO - add leading 0
3271 \def\bbl@replace@finish@iii#1{%
3272   \bbl@exp{\def\\#1####1####2####3{\the\toks@}}}
3273 \def\bbl@TG@@date{%
3274   \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace{}}%
3275   \bbl@replace\bbl@toreplace{[.]}{\BabelDateDot{}}%
3276   \bbl@replace\bbl@toreplace{[d]}{\BabelDated{####3}}%
3277   \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{####3}}%
3278   \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{####2}}%
3279   \bbl@replace\bbl@toreplace{[MM]}{\BabelDateMM{####2}}%
3280   \bbl@replace\bbl@toreplace{[MMMM]}{\BabelDateMMMM{####2}}%
3281   \bbl@replace\bbl@toreplace{[y]}{\BabelDatey{####1}}%
3282   \bbl@replace\bbl@toreplace{[yy]}{\BabelDateyy{####1}}%
3283   \bbl@replace\bbl@toreplace{[yyyy]}{\BabelDateyyyy{####1}}%
3284   \bbl@replace\bbl@toreplace{[y|}{\bbl@datecntr[####1|}%
3285   \bbl@replace\bbl@toreplace{[m|}{\bbl@datecntr[####2|}%
3286   \bbl@replace\bbl@toreplace{[d|}{\bbl@datecntr[####3|}%
3287   \bbl@replace@finish@iii\bbl@toreplace}
3288 \def\bbl@datecntr{\expandafter\bbl@xdatecntr\expandafter}
3289 \def\bbl@xdatecntr[#1|#2]{\localenumeral{#2}{#1}}
```

Transforms.

```
3290 \let\bbl@release@transforms\@empty
3291 \bbl@csarg\let{inikv@transforms.prehyphenation}\bbl@inikv
3292 \bbl@csarg\let{inikv@transforms.posthyphenation}\bbl@inikv
3293 \def\bbl@transforms@aux#1#2#3#4,#5\relax{%
3294   #1[#2]{#3}{#4}{#5}}
3295 \begingroup %  A hack. TODO. Don't require an specific order
3296   \catcode`\%=12
3297   \catcode`\&=14
3298   \gdef\bbl@transforms#1#2#3{&%
3299     \directlua{
3300       local str = [==[#2]==]
```

```
3301        str = str:gsub('%.%d+%.%d+$', '')
3302        token.set_macro('babeltempa', str)
3303      }&%
3304    \def\babeltempc{}&%
3305    \bbl@xin@{,\babeltempa,}{,\bbl@KVP@transforms,}&%
3306    \ifin@\else
3307      \bbl@xin@{:\babeltempa,}{,\bbl@KVP@transforms,}&%
3308    \fi
3309    \ifin@
3310      \bbl@foreach\bbl@KVP@transforms{&%
3311        \bbl@xin@{:\babeltempa,}{,##1,}&%
3312        \ifin@  &% font:font:transform syntax
3313          \directlua{
3314            local t = {}
3315            for m in string.gmatch('##1'..':', '(.-):') do
3316              table.insert(t, m)
3317            end
3318            table.remove(t)
3319            token.set_macro('babeltempc', ',fonts=' .. table.concat(t, ' '))
3320          }&%
3321        \fi}&%
3322      \in@{.0$}{#2$}&%
3323      \ifin@
3324        \directlua{&% (\attribute) syntax
3325          local str = string.match([[\bbl@KVP@transforms]],
3326                        '%(([^%(]-)%)[^%)]-\babeltempa')
3327          if str == nil then
3328            token.set_macro('babeltempb', '')
3329          else
3330            token.set_macro('babeltempb', ',attribute=' .. str)
3331          end
3332        }&%
3333      \toks@{#3}&%
3334      \bbl@exp{&%
3335        \\\g@addto@macro\\\bbl@release@transforms{&%
3336          \relax  &% Closes previous \bbl@transforms@aux
3337          \\\bbl@transforms@aux
3338            \\#1{label=\babeltempa\babeltempb\babeltempc}&%
3339              {\languagename}{\the\toks@}}}&%
3340      \else
3341        \g@addto@macro\bbl@release@transforms{, {#3}}&%
3342      \fi
3343    \fi}
3344 \endgroup
```

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

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

```
3361              \ifx\AtBeginDocument\@notprerr
3362                \expandafter\@secondoftwo   % to execute right now
3363              \fi
3364              \AtBeginDocument{%
3365                \bbl@patchfont{\bbl@xenohyph}%
3366                \expandafter\select@language\expandafter{\languagename}}%
3367        \fi}}%
3368   \fi
3369   \bbl@csarg\bbl@toglobal{lsys@#1}}
3370 \def\bbl@xenohyph@d{%
3371   \bbl@ifset{bbl@prehc@\languagename}%
3372     {\ifnum\hyphenchar\font=\defaulthyphenchar
3373        \iffontchar\font\bbl@cl{prehc}\relax
3374          \hyphenchar\font\bbl@cl{prehc}\relax
3375        \else\iffontchar\font"200B
3376          \hyphenchar\font"200B
3377        \else
3378          \bbl@warning
3379            {Neither 0 nor ZERO WIDTH SPACE are available\\%
3380             in the current font, and therefore the hyphen\\%
3381             will be printed. Try changing the fontspec's\\%
3382             'HyphenChar' to another value, but be aware\\%
3383             this setting is not safe (see the manual).\\%
3384             Reported}%
3385          \hyphenchar\font\defaulthyphenchar
3386        \fi\fi
3387     \fi}%
3388     {\hyphenchar\font\defaulthyphenchar}}
3389   % \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).

```
3390 \def\bbl@load@info#1{%
3391   \def\BabelBeforeIni##1##2{%
3392     \begingroup
3393       \bbl@read@ini{##1}0%
3394       \endinput           % babel- .tex may contain onlypreamble's
3395     \endgroup}%            boxed, to avoid extra spaces:
3396   {\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.

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

132

```
3416        \\\else\\\ifx3########1#4%
3417        \\\else\\\ifx4########1#5%
3418        \\\else\\\ifx5########1##1%
3419        \\\else\\\ifx6########1##2%
3420        \\\else\\\ifx7########1##3%
3421        \\\else\\\ifx8########1##4%
3422        \\\else\\\ifx9########1##5%
3423        \\\else########1%
3424        \\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi
3425        \\\expandafter\<bbl@digits@\languagename>%
3426      \\\fi}}}%
3427    \bbl@tempa}
```

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

```
3428 \def\bbl@buildifcase#1 {% Returns \bbl@tempa, requires \toks@={}
3429   \ifx\\#1%              % \\ before, in case #1 is multiletter
3430     \bbl@exp{%
3431       \def\\\bbl@tempa####1{%
3432         \<ifcase>####1\space\the\toks@\<else>\\\@ctrerr\<fi>}}%
3433   \else
3434     \toks@\expandafter{\the\toks@\or #1}%
3435     \expandafter\bbl@buildifcase
3436   \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).

```
3437 \newcommand\localenumeral[2]{\bbl@cs{cntr@#1@\languagename}{#2}}
3438 \def\bbl@localecntr#1#2{\localenumeral{#2}{#1}}
3439 \newcommand\localecounter[2]{%
3440   \expandafter\bbl@localecntr
3441   \expandafter{\number\csname c@#2\endcsname}{#1}}
3442 \def\bbl@alphnumeral#1#2{%
3443   \expandafter\bbl@alphnumeral@i\number#2 76543210\@@{#1}}
3444 \def\bbl@alphnumeral@i#1#2#3#4#5#6#7#8\@@#9{%
3445   \ifcase\@car#8\@nil\or    % Currenty <10000, but prepared for bigger
3446     \bbl@alphnumeral@ii{#9}000000#1\or
3447     \bbl@alphnumeral@ii{#9}00000#1#2\or
3448     \bbl@alphnumeral@ii{#9}0000#1#2#3\or
3449     \bbl@alphnumeral@ii{#9}000#1#2#3#4\else
3450     \bbl@alphnum@invalid{>9999}%
3451   \fi}
3452 \def\bbl@alphnumeral@ii#1#2#3#4#5#6#7#8{%
3453   \bbl@ifunset{bbl@cntr@#1.F.\number#5#6#7#8@\languagename}%
3454     {\bbl@cs{cntr@#1.4@\languagename}#5%
3455      \bbl@cs{cntr@#1.3@\languagename}#6%
3456      \bbl@cs{cntr@#1.2@\languagename}#7%
3457      \bbl@cs{cntr@#1.1@\languagename}#8%
3458      \ifnum#6#7#8>\z@ % TODO. An ad hoc rule for Greek. Ugly.
3459        \bbl@ifunset{bbl@cntr@#1.S.321@\languagename}{}%
3460          {\bbl@cs{cntr@#1.S.321@\languagename}}%
3461      \fi}%
3462     {\bbl@cs{cntr@#1.F.\number#5#6#7#8@\languagename}}}
3463 \def\bbl@alphnum@invalid#1{%
3464   \bbl@error{Alphabetic numeral too large (#1)}%
3465     {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.

```
3466 \def\bbl@localeinfo#1#2{%
3467   \bbl@ifunset{bbl@info@#2}{#1}%
3468     {\bbl@ifunset{bbl@\csname bbl@info@#2\endcsname @\languagename}{#1}%
```

```
3469        {\bbl@cs{\csname bbl@info@#2\endcsname @\languagename}}}}
3470 \newcommand\localeinfo[1]{%
3471  \ifx*#1\@empty   % TODO. A bit hackish to make it expandable.
3472    \bbl@afterelse\bbl@localeinfo{}%
3473  \else
3474    \bbl@localeinfo
3475      {\bbl@error{I've found no info for the current locale.\\%
3476                  The corresponding ini file has not been loaded\\%
3477                  Perhaps it doesn't exist}%
3478                 {See the manual for details.}}%
3479      {#1}%
3480  \fi}
3481 % \@namedef{bbl@info@name.locale}{lcname}
3482 \@namedef{bbl@info@tag.ini}{lini}
3483 \@namedef{bbl@info@name.english}{elname}
3484 \@namedef{bbl@info@name.opentype}{lname}
3485 \@namedef{bbl@info@tag.bcp47}{tbcp}
3486 \@namedef{bbl@info@language.tag.bcp47}{lbcp}
3487 \@namedef{bbl@info@tag.opentype}{lotf}
3488 \@namedef{bbl@info@script.name}{esname}
3489 \@namedef{bbl@info@script.name.opentype}{sname}
3490 \@namedef{bbl@info@script.tag.bcp47}{sbcp}
3491 \@namedef{bbl@info@script.tag.opentype}{sotf}
3492 \@namedef{bbl@info@region.tag.bcp47}{rbcp}
3493 \@namedef{bbl@info@variant.tag.bcp47}{vbcp}
3494 % Extensions are dealt with in a special way
3495 % Now, an internal \LaTeX{} macro:
3496 \providecommand\BCPdata[1]{\localeinfo*{#1.tag.bcp47}}
```

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

```
3497 ⟨⟨*More package options⟩⟩ ≡
3498 \DeclareOption{ensureinfo=off}{}
3499 ⟨⟨/More package options⟩⟩
3500 %
3501 \let\bbl@ensureinfo\@gobble
3502 \newcommand\BabelEnsureInfo{%
3503  \ifx\InputIfFileExists\@undefined\else
3504    \def\bbl@ensureinfo##1{%
3505      \bbl@ifunset{bbl@lname@##1}{\bbl@load@info{##1}}{}}%
3506  \fi
3507  \bbl@foreach\bbl@loaded{{%
3508    \def\languagename{##1}%
3509    \bbl@ensureinfo{##1}}}}
3510 \@ifpackagewith{babel}{ensureinfo=off}{}%
3511  {\AtEndOfPackage{% Test for plain.
3512    \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.

```
3513 \newcommand\getlocaleproperty{%
3514  \@ifstar\bbl@getproperty@s\bbl@getproperty@x}
3515 \def\bbl@getproperty@s#1#2#3{%
3516  \let#1\relax
3517  \def\bbl@elt##1##2##3{%
3518    \bbl@ifsamestring{##1/##2}{#3}%
3519      {\providecommand#1{##3}%
3520       \def\bbl@elt####1####2####3{}}%
3521      {}}%
3522  \bbl@cs{inidata@#2}}%
3523 \def\bbl@getproperty@x#1#2#3{%
3524  \bbl@getproperty@s{#1}{#2}{#3}%
3525  \ifx#1\relax
3526    \bbl@error
```

```
3527        {Unknown key for locale '#2':\\%
3528          #3\\%
3529          \string#1 will be set to \relax}%
3530        {Perhaps you misspelled it.}%
3531    \fi}
3532 \let\bbl@ini@loaded\@empty
3533 \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.

```
3534 \newcommand\babeladjust[1]{%  TODO. Error handling.
3535    \bbl@forkv{#1}{%
3536      \bbl@ifunset{bbl@ADJ@##1@##2}%
3537        {\bbl@cs{ADJ@##1}{##2}}%
3538        {\bbl@cs{ADJ@##1@##2}}}}
3539 %
3540 \def\bbl@adjust@lua#1#2{%
3541    \ifvmode
3542      \ifnum\currentgrouplevel=\z@
3543        \directlua{ Babel.#2 }%
3544        \expandafter\expandafter\expandafter\@gobble
3545      \fi
3546    \fi
3547    {\bbl@error   % The error is gobbled if everything went ok.
3548      {Currently, #1 related features can be adjusted only\\%
3549        in the main vertical list.}%
3550      {Maybe things change in the future, but this is what it is.}}}
3551 \@namedef{bbl@ADJ@bidi.mirroring@on}{%
3552    \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
3553 \@namedef{bbl@ADJ@bidi.mirroring@off}{%
3554    \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
3555 \@namedef{bbl@ADJ@bidi.text@on}{%
3556    \bbl@adjust@lua{bidi}{bidi_enabled=true}}
3557 \@namedef{bbl@ADJ@bidi.text@off}{%
3558    \bbl@adjust@lua{bidi}{bidi_enabled=false}}
3559 \@namedef{bbl@ADJ@bidi.math@on}{%
3560    \let\bbl@noamsmath\@empty}
3561 \@namedef{bbl@ADJ@bidi.math@off}{%
3562    \let\bbl@noamsmath\relax}
3563 \@namedef{bbl@ADJ@bidi.mapdigits@on}{%
3564    \bbl@adjust@lua{bidi}{digits_mapped=true}}
3565 \@namedef{bbl@ADJ@bidi.mapdigits@off}{%
3566    \bbl@adjust@lua{bidi}{digits_mapped=false}}
3567 %
3568 \@namedef{bbl@ADJ@linebreak.sea@on}{%
3569    \bbl@adjust@lua{linebreak}{sea_enabled=true}}
3570 \@namedef{bbl@ADJ@linebreak.sea@off}{%
3571    \bbl@adjust@lua{linebreak}{sea_enabled=false}}
3572 \@namedef{bbl@ADJ@linebreak.cjk@on}{%
3573    \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
3574 \@namedef{bbl@ADJ@linebreak.cjk@off}{%
3575    \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
3576 \@namedef{bbl@ADJ@justify.arabic@on}{%
3577    \bbl@adjust@lua{linebreak}{arabic.justify_enabled=true}}
3578 \@namedef{bbl@ADJ@justify.arabic@off}{%
3579    \bbl@adjust@lua{linebreak}{arabic.justify_enabled=false}}
3580 %
3581 \def\bbl@adjust@layout#1{%
3582    \ifvmode
3583      #1%
3584      \expandafter\@gobble
```

```
3585  \fi
3586  {\bbl@error    % The error is gobbled if everything went ok.
3587    {Currently, layout related features can be adjusted only\\%
3588     in vertical mode.}%
3589    {Maybe things change in the future, but this is what it is.}}}
3590 \@namedef{bbl@ADJ@layout.tabular@on}{%
3591  \ifnum\bbl@tabular@mode=\tw@
3592    \bbl@adjust@layout{\let\@tabular\bbl@NL@@tabular}%
3593  \else
3594    \chardef\bbl@tabular@mode\@ne
3595  \fi}
3596 \@namedef{bbl@ADJ@layout.tabular@off}{%
3597  \ifnum\bbl@tabular@mode=\tw@
3598    \bbl@adjust@layout{\let\@tabular\bbl@OL@@tabular}%
3599  \else
3600    \chardef\bbl@tabular@mode\z@
3601  \fi}
3602 \@namedef{bbl@ADJ@layout.lists@on}{%
3603  \bbl@adjust@layout{\let\list\bbl@NL@list}}
3604 \@namedef{bbl@ADJ@layout.lists@off}{%
3605  \bbl@adjust@layout{\let\list\bbl@OL@list}}
3606 %
3607 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
3608  \bbl@bcpallowedtrue}
3609 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
3610  \bbl@bcpallowedfalse}
3611 \@namedef{bbl@ADJ@autoload.bcp47.prefix}#1{%
3612  \def\bbl@bcp@prefix{#1}}
3613 \def\bbl@bcp@prefix{bcp47-}
3614 \@namedef{bbl@ADJ@autoload.options}#1{%
3615  \def\bbl@autoload@options{#1}}
3616 \let\bbl@autoload@bcpoptions\@empty
3617 \@namedef{bbl@ADJ@autoload.bcp47.options}#1{%
3618  \def\bbl@autoload@bcpoptions{#1}}
3619 \newif\ifbbl@bcptoname
3620 \@namedef{bbl@ADJ@bcp47.toname@on}{%
3621  \bbl@bcptonametrue
3622  \BabelEnsureInfo}
3623 \@namedef{bbl@ADJ@bcp47.toname@off}{%
3624  \bbl@bcptonamefalse}
3625 \@namedef{bbl@ADJ@prehyphenation.disable@nohyphenation}{%
3626  \directlua{ Babel.ignore_pre_char = function(node)
3627    return (node.lang == \the\csname l@nohyphenation\endcsname)
3628   end }}
3629 \@namedef{bbl@ADJ@prehyphenation.disable@off}{%
3630  \directlua{ Babel.ignore_pre_char = function(node)
3631    return false
3632   end }}
3633 \@namedef{bbl@ADJ@select.write@shift}{%
3634  \let\bbl@restorelastskip\relax
3635  \def\bbl@savelastskip{%
3636    \let\bbl@restorelastskip\relax
3637    \ifvmode
3638      \ifdim\lastskip=\z@
3639        \let\bbl@restorelastskip\nobreak
3640      \else
3641        \bbl@exp{%
3642          \def\\\bbl@restorelastskip{%
3643            \skip@=\the\lastskip
3644            \\\nobreak \vskip-\skip@ \vskip\skip@}}%
3645      \fi
3646    \fi}}
3647 \@namedef{bbl@ADJ@select.write@keep}{%
```

```
3648    \let\bbl@restorelastskip\relax
3649    \let\bbl@savelastskip\relax}
3650 \@namedef{bbl@ADJ@select.write@omit}{%
3651    \AddBabelHook{babel-select}{beforestart}{%
3652      \expandafter\babel@aux\expandafter{\bbl@main@language}{}}%
3653    \let\bbl@restorelastskip\relax
3654    \def\bbl@savelastskip##1\bbl@restorelastskip{}}
3655 \@namedef{bbl@ADJ@select.encoding@off}{%
3656    \let\bbl@encoding@select@off\@empty}
```

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

```
3657 \ifx\directlua\@undefined\else
3658   \ifx\bbl@luapatterns\@undefined
3659     \input luababel.def
3660   \fi
3661 \fi
```

Continue with LaTeX.

```
3662 ⟨/package | core⟩
3663 ⟨∗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.

```
3664 ⟨⟨∗More package options⟩⟩ ≡
3665 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
3666 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
3667 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
3668 \DeclareOption{safe=refbib}{\def\bbl@opt@safe{BR}}
3669 \DeclareOption{safe=bibref}{\def\bbl@opt@safe{BR}}
3670 ⟨⟨/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.

```
3671 \bbl@trace{Cross referencing macros}
3672 \ifx\bbl@opt@safe\@empty\else % ie, if 'ref' and/or 'bib'
3673   \def\@newl@bel#1#2#3{%
3674     {\@safe@activestrue
3675      \bbl@ifunset{#1@#2}%
3676        \relax
3677        {\gdef\@multiplelabels{%
3678           \@latex@warning@no@line{There were multiply-defined labels}}%
3679         \@latex@warning@no@line{Label `#2' multiply defined}}%
3680      \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.

```
3681    \CheckCommand*\@testdef[3]{%
3682      \def\reserved@a{#3}%
3683      \expandafter\ifx\csname#1@#2\endcsname\reserved@a
3684      \else
3685        \@tempswatrue
3686      \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.

```
3687   \def\@testdef#1#2#3{%  TODO. With @samestring?
3688     \@safe@activestrue
3689     \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
3690     \def\bbl@tempb{#3}%
3691     \@safe@activesfalse
3692     \ifx\bbl@tempa\relax
3693     \else
3694       \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
3695     \fi
3696     \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
3697     \ifx\bbl@tempa\bbl@tempb
3698     \else
3699       \@tempswatrue
3700     \fi}
3701 \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.

```
3702 \bbl@xin@{R}\bbl@opt@safe
3703 \ifin@
3704   \edef\bbl@tempc{\expandafter\string\csname ref code\endcsname}%
3705   \bbl@xin@{\expandafter\strip@prefix\meaning\bbl@tempc}%
3706     {\expandafter\strip@prefix\meaning\ref}%
3707   \ifin@
3708     \bbl@redefine\@kernel@ref#1{%
3709       \@safe@activestrue\org@@kernel@ref{#1}\@safe@activesfalse}
3710     \bbl@redefine\@kernel@pageref#1{%
3711       \@safe@activestrue\org@@kernel@pageref{#1}\@safe@activesfalse}
3712     \bbl@redefine\@kernel@sref#1{%
3713       \@safe@activestrue\org@@kernel@sref{#1}\@safe@activesfalse}
3714     \bbl@redefine\@kernel@spageref#1{%
3715       \@safe@activestrue\org@@kernel@spageref{#1}\@safe@activesfalse}
3716   \else
3717     \bbl@redefinerobust\ref#1{%
3718       \@safe@activestrue\org@ref{#1}\@safe@activesfalse}
3719     \bbl@redefinerobust\pageref#1{%
3720       \@safe@activestrue\org@pageref{#1}\@safe@activesfalse}
3721   \fi
3722 \else
3723   \let\org@ref\ref
3724   \let\org@pageref\pageref
3725 \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.

```
3726 \bbl@xin@{B}\bbl@opt@safe
3727 \ifin@
3728   \bbl@redefine\@citex[#1]#2{%
3729     \@safe@activestrue\edef\@tempa{#2}\@safe@activesfalse
3730     \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.

```
3731  \AtBeginDocument{%
3732    \@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.)

```
3733    \def\@citex[#1][#2]#3{%
3734      \@safe@activestrue\edef\@tempa{#3}\@safe@activesfalse
3735      \org@@citex[#1][#2]{\@tempa}}%
3736    }{}}
```

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

```
3737  \AtBeginDocument{%
3738    \@ifpackageloaded{cite}{%
3739      \def\@citex[#1]#2{%
3740        \@safe@activestrue\org@@citex[#1]{#2}\@safe@activesfalse}%
3741    }{}}
```

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

```
3742  \bbl@redefine\nocite#1{%
3743    \@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.

```
3744  \bbl@redefine\bibcite{%
3745    \bbl@cite@choice
3746    \bibcite}
```

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

```
3747  \def\bbl@bibcite#1#2{%
3748    \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.

```
3749  \def\bbl@cite@choice{%
3750    \global\let\bibcite\bbl@bibcite
3751    \@ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{}%
3752    \@ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{}%
3753    \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.

```
3754  \AtBeginDocument{\bbl@cite@choice}
```

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

```
3755  \bbl@redefine\@bibitem#1{%
3756    \@safe@activestrue\org@@bibitem{#1}\@safe@activesfalse}
3757 \else
3758   \let\org@nocite\nocite
3759   \let\org@@citex\@citex
3760   \let\org@bibcite\bibcite
3761   \let\org@@bibitem\@bibitem
3762 \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.

```
3763 \bbl@trace{Marks}
3764 \IfBabelLayout{sectioning}
3765   {\ifx\bbl@opt@headfoot\@nnil
3766     \g@addto@macro\@resetactivechars{%
3767       \set@typeset@protect
3768       \expandafter\select@language@x\expandafter{\bbl@main@language}%
3769       \let\protect\noexpand
3770       \ifcase\bbl@bidimode\else % Only with bidi. See also above
3771         \edef\thepage{%
3772           \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
3773       \fi}%
3774   \fi}
3775 {\ifbbl@single\else
3776   \bbl@ifunset{markright }\bbl@redefine\bbl@redefinerobust
3777   \markright#1{%
3778     \bbl@ifblank{#1}%
3779       {\org@markright{}}%
3780       {\toks@{#1}%
3781        \bbl@exp{%
3782          \\\org@markright{\\\protect\\\foreignlanguage{\languagename}%
3783            {\\\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.)

```
3784     \ifx\@mkboth\markboth
3785       \def\bbl@tempc{\let\@mkboth\markboth}%
3786     \else
3787       \def\bbl@tempc{}%
3788     \fi
3789     \bbl@ifunset{markboth }\bbl@redefine\bbl@redefinerobust
3790     \markboth#1#2{%
3791       \protected@edef\bbl@tempb##1{%
3792         \protect\foreignlanguage
3793         {\languagename}{\protect\bbl@restore@actives##1}}%
3794       \bbl@ifblank{#1}%
3795         {\toks@{}}%
3796         {\toks@\expandafter{\bbl@tempb{#1}}}%
3797       \bbl@ifblank{#2}%
3798         {\@temptokena{}}%
3799         {\@temptokena\expandafter{\bbl@tempb{#2}}}%
3800       \bbl@exp{\\\org@markboth{\the\toks@}{\the\@temptokena}}%
3801       \bbl@tempc
3802   \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.

```
3803 \bbl@trace{Preventing clashes with other packages}
3804 \ifx\org@ref\@undefined\else
3805   \bbl@xin@{R}\bbl@opt@safe
3806   \ifin@
3807     \AtBeginDocument{%
3808       \@ifpackageloaded{ifthen}{%
3809         \bbl@redefine@long\ifthenelse#1#2#3{%
3810           \let\bbl@temp@pref\pageref
3811           \let\pageref\org@pageref
3812           \let\bbl@temp@ref\ref
3813           \let\ref\org@ref
3814           \@safe@activestrue
3815           \org@ifthenelse{#1}%
3816             {\let\pageref\bbl@temp@pref
3817              \let\ref\bbl@temp@ref
3818              \@safe@activesfalse
3819              #2}%
3820             {\let\pageref\bbl@temp@pref
3821              \let\ref\bbl@temp@ref
3822              \@safe@activesfalse
3823              #3}%
3824         }%
3825       }{}%
3826     }
3827 \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`.

```
3828   \AtBeginDocument{%
3829     \@ifpackageloaded{varioref}{%
3830       \bbl@redefine\@@vpageref#1[#2]#3{%
3831         \@safe@activestrue
3832         \org@@@vpageref{#1}[#2]{#3}%
3833         \@safe@activesfalse}%
3834       \bbl@redefine\vrefpagenum#1#2{%
3835         \@safe@activestrue
3836         \org@vrefpagenum{#1}{#2}%
3837         \@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.

```
3838       \expandafter\def\csname Ref \endcsname#1{%
3839         \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
3840       }{}%
```

141

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.

```
3843 \AtEndOfPackage{%
3844   \AtBeginDocument{%
3845     \@ifpackageloaded{hhline}%
3846       {\expandafter\ifx\csname normal@char\string:\endcsname\relax
3847        \else
3848          \makeatletter
3849          \def\@currname{hhline}\input{hhline.sty}\makeatother
3850        \fi}%
3851       {}}}
```

\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.

```
3852 \def\substitutefontfamily#1#2#3{%
3853   \lowercase{\immediate\openout15=#1#2.fd\relax}%
3854   \immediate\write15{%
3855     \string\ProvidesFile{#1#2.fd}%
3856     [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
3857      \space generated font description file]^^J
3858     \string\DeclareFontFamily{#1}{#2}{}^^J
3859     \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{}^^J
3860     \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{}^^J
3861     \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{}^^J
3862     \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{}^^J
3863     \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{}^^J
3864     \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{}^^J
3865     \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{}^^J
3866     \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{}^^J
3867     }%
3868   \closeout15
3869   }
3870 \@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

```
3871 \bbl@trace{Encoding and fonts}
3872 \newcommand\BabelNonASCII{LGR,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU}
3873 \newcommand\BabelNonText{TS1,T3,TS3}
3874 \let\org@TeX\TeX
3875 \let\org@LaTeX\LaTeX
3876 \let\ensureascii\@firstofone
3877 \AtBeginDocument{%
3878   \def\@elt#1{,#1,}%
3879   \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3880   \let\@elt\relax
```

```
3881    \let\bbl@tempb\@empty
3882    \def\bbl@tempc{OT1}%
3883    \bbl@foreach\BabelNonASCII{% LGR loaded in a non-standard way
3884      \bbl@ifunset{T@#1}{}{\def\bbl@tempb{#1}}}%
3885    \bbl@foreach\bbl@tempa{%
3886      \bbl@xin@{#1}{\BabelNonASCII}%
3887      \ifin@
3888        \def\bbl@tempb{#1}% Store last non-ascii
3889      \else\bbl@xin@{#1}{\BabelNonText}% Pass
3890        \ifin@\else
3891          \def\bbl@tempc{#1}% Store last ascii
3892        \fi
3893      \fi}%
3894    \ifx\bbl@tempb\@empty\else
3895      \bbl@xin@{,\cf@encoding,}{,\BabelNonASCII,\BabelNonText,}%
3896      \ifin@\else
3897        \edef\bbl@tempc{\cf@encoding}% The default if ascii wins
3898      \fi
3899      \edef\ensureascii#1{%
3900        {\noexpand\fontencoding{\bbl@tempc}\noexpand\selectfont#1}}%
3901      \DeclareTextCommandDefault{\TeX}{\ensureascii{\org@TeX}}%
3902      \DeclareTextCommandDefault{\LaTeX}{\ensureascii{\org@LaTeX}}%
3903    \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.

```
3904 \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.

```
3905 \AtBeginDocument{%
3906   \@ifpackageloaded{fontspec}%
3907     {\xdef\latinencoding{%
3908        \ifx\UTFencname\@undefined
3909          EU\ifcase\bbl@engine\or2\or1\fi
3910        \else
3911          \UTFencname
3912        \fi}}%
3913     {\gdef\latinencoding{OT1}%
3914      \ifx\cf@encoding\bbl@t@one
3915        \xdef\latinencoding{\bbl@t@one}%
3916      \else
3917        \def\@elt#1{,#1,}%
3918        \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3919        \let\@elt\relax
3920        \bbl@xin@{,T1,}\bbl@tempa
3921        \ifin@
3922          \xdef\latinencoding{\bbl@t@one}%
3923        \fi
3924      \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.

```
3925 \DeclareRobustCommand{\latintext}{%
3926   \fontencoding{\latinencoding}\selectfont
3927   \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.

```
3928 \ifx\@undefined\DeclareTextFontCommand
3929   \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
3930 \else
3931   \DeclareTextFontCommand{\textlatin}{\latintext}
3932 \fi
```

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

```
3933 \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.

```
3934 \bbl@trace{Loading basic (internal) bidi support}
3935 \ifodd\bbl@engine
3936 \else % TODO. Move to txtbabel
3937   \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
3938     \bbl@error
3939       {The bidi method 'basic' is available only in\\%
3940        luatex. I'll continue with 'bidi=default', so\\%
3941        expect wrong results}%
3942       {See the manual for further details.}%
3943     \let\bbl@beforeforeign\leavevmode
3944     \AtEndOfPackage{%
3945       \EnableBabelHook{babel-bidi}%
3946       \bbl@xebidipar}
3947   \fi\fi
3948   \def\bbl@loadxebidi#1{%
3949     \ifx\RTLfootnotetext\@undefined
3950       \AtEndOfPackage{%
3951         \EnableBabelHook{babel-bidi}%
3952         \bbl@loadfontspec % bidi needs fontspec
3953         \usepackage#1{bidi}}%
3954     \fi}
3955   \ifnum\bbl@bidimode>200
3956     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
3957       \bbl@tentative{bidi=bidi}
3958       \bbl@loadxebidi{}
3959     \or
3960       \bbl@loadxebidi{[rldocument]}
3961     \or
```

```
3962        \bbl@loadxebidi{}
3963      \fi
3964    \fi
3965  \fi
3966  % TODO? Separate:
3967  \ifnum\bbl@bidimode=\@ne
3968    \let\bbl@beforeforeign\leavevmode
3969    \ifodd\bbl@engine
3970      \newattribute\bbl@attr@dir
3971      \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
3972      \bbl@exp{\output{\bodydir\pagedir\the\output}}
3973    \fi
3974    \AtEndOfPackage{%
3975      \EnableBabelHook{babel-bidi}%
3976      \ifodd\bbl@engine\else
3977        \bbl@xebidipar
3978      \fi}
3979  \fi
```

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

```
3980  \bbl@trace{Macros to switch the text direction}
3981  \def\bbl@alscripts{,Arabic,Syriac,Thaana,}
3982  \def\bbl@rscripts{% TODO. Base on codes ??
3983    ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
3984    Old Hungarian,Lydian,Mandaean,Manichaean,%
3985    Meroitic Cursive,Meroitic,Old North Arabian,%
3986    Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
3987    Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
3988    Old South Arabian,}%
3989  \def\bbl@provide@dirs#1{%
3990    \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
3991    \ifin@
3992      \global\bbl@csarg\chardef{wdir@#1}\@ne
3993      \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
3994      \ifin@
3995        \global\bbl@csarg\chardef{wdir@#1}\tw@  % useless in xetex
3996      \fi
3997    \else
3998      \global\bbl@csarg\chardef{wdir@#1}\z@
3999    \fi
4000    \ifodd\bbl@engine
4001      \bbl@csarg\ifcase{wdir@#1}%
4002        \directlua{ Babel.locale_props[\the\localeid].textdir = 'l' }%
4003      \or
4004        \directlua{ Babel.locale_props[\the\localeid].textdir = 'r' }%
4005      \or
4006        \directlua{ Babel.locale_props[\the\localeid].textdir = 'al' }%
4007      \fi
4008    \fi}
4009  \def\bbl@switchdir{%
4010    \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
4011    \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
4012    \bbl@exp{\\\bbl@setdirs\bbl@cl{wdir}}}
4013  \def\bbl@setdirs#1{% TODO - math
4014    \ifcase\bbl@select@type % TODO - strictly, not the right test
4015      \bbl@bodydir{#1}%
4016      \bbl@pardir{#1}% <- Must precede \bbl@textdir
4017    \fi
4018    \bbl@textdir{#1}}
4019  % TODO. Only if \bbl@bidimode > 0?:
4020  \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
4021  \DisableBabelHook{babel-bidi}
```

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

```
4022 \ifodd\bbl@engine   % luatex=1
4023 \else % pdftex=0, xetex=2
4024   \newcount\bbl@dirlevel
4025   \chardef\bbl@thetextdir\z@
4026   \chardef\bbl@thepardir\z@
4027   \def\bbl@textdir#1{%
4028     \ifcase#1\relax
4029       \chardef\bbl@thetextdir\z@
4030       \bbl@textdir@i\beginL\endL
4031     \else
4032       \chardef\bbl@thetextdir\@ne
4033       \bbl@textdir@i\beginR\endR
4034     \fi}
4035   \def\bbl@textdir@i#1#2{%
4036     \ifhmode
4037       \ifnum\currentgrouplevel>\z@
4038         \ifnum\currentgrouplevel=\bbl@dirlevel
4039           \bbl@error{Multiple bidi settings inside a group}%
4040             {I'll insert a new group, but expect wrong results.}%
4041           \bgroup\aftergroup#2\aftergroup\egroup
4042         \else
4043           \ifcase\currentgrouptype\or % 0 bottom
4044             \aftergroup#2% 1 simple {}
4045           \or
4046             \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
4047           \or
4048             \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
4049           \or\or\or % vbox vtop align
4050           \or
4051             \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
4052           \or\or\or\or\or\or % output math disc insert vcent mathchoice
4053           \or
4054             \aftergroup#2% 14 \begingroup
4055           \else
4056             \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
4057           \fi
4058         \fi
4059         \bbl@dirlevel\currentgrouplevel
4060       \fi
4061       #1%
4062     \fi}
4063   \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
4064   \let\bbl@bodydir\@gobble
4065   \let\bbl@pagedir\@gobble
4066   \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).

```
4067   \def\bbl@xebidipar{%
4068     \let\bbl@xebidipar\relax
4069     \TeXXeTstate\@ne
4070     \def\bbl@xeeverypar{%
4071       \ifcase\bbl@thepardir
4072         \ifcase\bbl@thetextdir\else\beginR\fi
4073       \else
4074         {\setbox\z@\lastbox\beginR\box\z@}%
4075       \fi}%
4076     \let\bbl@severypar\everypar
4077     \newtoks\everypar
4078     \everypar=\bbl@severypar
4079     \bbl@severypar{\bbl@xeeverypar\the\everypar}}
```

```
4080   \ifnum\bbl@bidimode>200
4081     \let\bbl@textdir@i\@gobbletwo
4082     \let\bbl@xebidipar\@empty
4083     \AddBabelHook{bidi}{foreign}{%
4084       \def\bbl@tempa{\def\BabelText####1}%
4085       \ifcase\bbl@thetextdir
4086         \expandafter\bbl@tempa\expandafter{\BabelText{\LR{##1}}}%
4087       \else
4088         \expandafter\bbl@tempa\expandafter{\BabelText{\RL{##1}}}%
4089       \fi}
4090     \def\bbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}
4091   \fi
4092 \fi
```

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

```
4093 \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}
4094 \AtBeginDocument{%
4095   \ifx\pdfstringdefDisableCommands\@undefined\else
4096     \ifx\pdfstringdefDisableCommands\relax\else
4097       \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
4098     \fi
4099   \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.

```
4100 \bbl@trace{Local Language Configuration}
4101 \ifx\loadlocalcfg\@undefined
4102   \@ifpackagewith{babel}{noconfigs}%
4103     {\let\loadlocalcfg\@gobble}%
4104     {\def\loadlocalcfg#1{%
4105       \InputIfFileExists{#1.cfg}%
4106         {\typeout{*************************************^^J%
4107                    * Local config file #1.cfg used^^J%
4108                    *}}%
4109         \@empty}}
4110 \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).

```
4111 \bbl@trace{Language options}
4112 \let\bbl@afterlang\relax
4113 \let\BabelModifiers\relax
4114 \let\bbl@loaded\@empty
4115 \def\bbl@load@language#1{%
4116   \InputIfFileExists{#1.ldf}%
4117     {\edef\bbl@loaded{\CurrentOption
4118         \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}%
4119      \expandafter\let\expandafter\bbl@afterlang
4120        \csname\CurrentOption.ldf-h@@k\endcsname
4121      \expandafter\let\expandafter\BabelModifiers
4122        \csname bbl@mod@\CurrentOption\endcsname}%
4123     {\bbl@error{%
4124       Unknown option '\CurrentOption'. Either you misspelled it\\%
4125       or the language definition file \CurrentOption.ldf was not found}{%
```

```
4126          Valid options are, among others: shorthands=, KeepShorthandsActive,\\%
4127          activeacute, activegrave, noconfigs, safe=, main=, math=\\%
4128          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.

```
4129 \def\bbl@try@load@lang#1#2#3{%
4130   \IfFileExists{\CurrentOption.ldf}%
4131     {\bbl@load@language{\CurrentOption}}%
4132     {#1\bbl@load@language{#2}#3}}
4133 %
4134 \DeclareOption{hebrew}{%
4135   \input{rlbabel.def}%
4136   \bbl@load@language{hebrew}}
4137 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}{}}
4138 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}{}}
4139 \DeclareOption{nynorsk}{\bbl@try@load@lang{}{norsk}{}}
4140 \DeclareOption{polutonikogreek}{%
4141   \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}
4142 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}{}}
4143 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}{}}
4144 \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.

```
4145 \ifx\bbl@opt@config\@nnil
4146   \@ifpackagewith{babel}{noconfigs}{}%
4147     {\InputIfFileExists{bblopts.cfg}%
4148       {\typeout{*************************************^^J%
4149               * Local config file bblopts.cfg used^^J%
4150               *}}%
4151      {}}%
4152 \else
4153   \InputIfFileExists{\bbl@opt@config.cfg}%
4154     {\typeout{*************************************^^J%
4155               * Local config file \bbl@opt@config.cfg used^^J%
4156               *}}%
4157    {\bbl@error{%
4158       Local config file '\bbl@opt@config.cfg' not found}{%
4159       Perhaps you misspelled it.}}%
4160 \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.

```
4161 \ifx\bbl@opt@main\@nnil
4162   \ifnum\bbl@iniflag>\z@  % if all ldf's: set implicitly, no main pass
4163     \let\bbl@tempb\@empty
4164     \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}%
4165     \bbl@foreach\bbl@tempa{\edef\bbl@tempb{#1,\bbl@tempb}}%
4166     \bbl@foreach\bbl@tempb{%     \bbl@tempb is a reversed list
4167       \ifx\bbl@opt@main\@nnil % ie, if not yet assigned
4168         \ifodd\bbl@iniflag % = *=
4169           \IfFileExists{babel-#1.tex}{\def\bbl@opt@main{#1}}{}%
4170         \else % n +=
4171           \IfFileExists{#1.ldf}{\def\bbl@opt@main{#1}}{}%
4172         \fi
```

148

```
4173        \fi}%
4174    \fi
4175 \else
4176    \bbl@info{Main language set with 'main='. Except if you have\\%
4177            problems, prefer the default mechanism for setting\\%
4178            the main language. Reported}
4179 \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).

```
4180 \ifx\bbl@opt@main\@nnil\else
4181    \bbl@ncarg\let\bbl@loadmain{ds@\bbl@opt@main}%
4182    \expandafter\let\csname ds@\bbl@opt@main\endcsname\relax
4183 \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.

```
4184 \bbl@foreach\bbl@language@opts{%
4185    \def\bbl@tempa{#1}%
4186    \ifx\bbl@tempa\bbl@opt@main\else
4187        \ifnum\bbl@iniflag<\tw@     % 0 ø (other = ldf)
4188            \bbl@ifunset{ds@#1}%
4189                {\DeclareOption{#1}{\bbl@load@language{#1}}}%
4190                {}%
4191        \else                      % + * (other = ini)
4192            \DeclareOption{#1}{%
4193                \bbl@ldfinit
4194                \babelprovide[import]{#1}%
4195                \bbl@afterldf{}}%
4196        \fi
4197    \fi}
4198 \bbl@foreach\@classoptionslist{%
4199    \def\bbl@tempa{#1}%
4200    \ifx\bbl@tempa\bbl@opt@main\else
4201        \ifnum\bbl@iniflag<\tw@     % 0 ø (other = ldf)
4202            \bbl@ifunset{ds@#1}%
4203                {\IfFileExists{#1.ldf}%
4204                    {\DeclareOption{#1}{\bbl@load@language{#1}}}%
4205                    {}}%
4206                {}%
4207        \else                      % + * (other = ini)
4208            \IfFileExists{babel-#1.tex}%
4209                {\DeclareOption{#1}{%
4210                    \bbl@ldfinit
4211                    \babelprovide[import]{#1}%
4212                    \bbl@afterldf{}}}%
4213                {}%
4214        \fi
4215    \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):

```
4216 \def\AfterBabelLanguage#1{%
4217    \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang}{}}
4218 \DeclareOption*{}
4219 \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.

```
4220 \bbl@trace{Option 'main'}
4221 \ifx\bbl@opt@main\@nnil
4222   \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}
4223   \let\bbl@tempc\@empty
4224   \edef\bbl@templ{,\bbl@loaded,}
4225   \edef\bbl@templ{\expandafter\strip@prefix\meaning\bbl@templ}
4226   \bbl@for\bbl@tempb\bbl@tempa{%
4227     \edef\bbl@tempd{,\bbl@tempb,}%
4228     \edef\bbl@tempd{\expandafter\strip@prefix\meaning\bbl@tempd}%
4229     \bbl@xin@{\bbl@tempd}{\bbl@templ}%
4230     \ifin@\edef\bbl@tempc{\bbl@tempb}\fi}
4231   \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
4232   \expandafter\bbl@tempa\bbl@loaded,\@nnil
4233   \ifx\bbl@tempb\bbl@tempc\else
4234     \bbl@warning{%
4235       Last declared language option is '\bbl@tempc',\\%
4236       but the last processed one was '\bbl@tempb'.\\%
4237       The main language can't be set as both a global\\%
4238       and a package option. Use 'main=\bbl@tempc' as\\%
4239       option. Reported}
4240   \fi
4241 \else
4242   \ifodd\bbl@iniflag  % case 1,3 (main is ini)
4243     \bbl@ldfinit
4244     \let\CurrentOption\bbl@opt@main
4245     \bbl@exp{%  \bbl@opt@provide = empty if *
4246        \\\babelprovide[\bbl@opt@provide,import,main]{\bbl@opt@main}}%
4247     \bbl@afterldf{}
4248     \DeclareOption{\bbl@opt@main}{}
4249   \else % case 0,2 (main is ldf)
4250     \ifx\bbl@loadmain\relax
4251       \DeclareOption{\bbl@opt@main}{\bbl@load@language{\bbl@opt@main}}
4252     \else
4253       \DeclareOption{\bbl@opt@main}{\bbl@loadmain}
4254     \fi
4255     \ExecuteOptions{\bbl@opt@main}
4256     \@namedef{ds@\bbl@opt@main}{}%
4257   \fi
4258   \DeclareOption*{}
4259   \ProcessOptions*
4260 \fi
4261 \def\AfterBabelLanguage{%
4262   \bbl@error
4263     {Too late for \string\AfterBabelLanguage}%
4264     {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.

```
4265 \ifx\bbl@main@language\@undefined
4266   \bbl@info{%
4267     You haven't specified a language as a class or package\\%
4268     option. I'll load 'nil'. Reported}
4269     \bbl@load@language{nil}
4270 \fi
4271 ⟨/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

```
4272 ⟨∗kernel⟩
4273 \let\bbl@onlyswitch\@empty
4274 \input babel.def
4275 \let\bbl@onlyswitch\@undefined
4276 ⟨/kernel⟩
4277 ⟨∗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.

```
4278 ⟨⟨Make sure ProvidesFile is defined⟩⟩
4279 \ProvidesFile{hyphen.cfg}[⟨⟨date⟩⟩ ⟨⟨version⟩⟩ Babel hyphens]
4280 \xdef\bbl@format{\jobname}
4281 \def\bbl@version{⟨⟨version⟩⟩}
4282 \def\bbl@date{⟨⟨date⟩⟩}
4283 \ifx\AtBeginDocument\@undefined
4284   \def\@empty{}
4285 \fi
4286 ⟨⟨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.

```
4287 \def\process@line#1#2 #3 #4 {%
4288   \ifx=#1%
4289     \process@synonym{#2}%
4290   \else
4291     \process@language{#1#2}{#3}{#4}%
4292   \fi
4293   \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.

```
4294 \toks@{}
4295 \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.

```
4296 \def\process@synonym#1{%
4297   \ifnum\last@language=\m@ne
4298     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4299   \else
4300     \expandafter\chardef\csname l@#1\endcsname\last@language
4301     \wlog{\string\l@#1=\string\language\the\last@language}%
4302     \expandafter\let\csname #1hyphenmins\expandafter\endcsname
4303       \csname\languagename hyphenmins\endcsname
4304     \let\bbl@elt\relax
```

151

```
4305        \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}{}{}}%
4306    \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.

```
4307 \def\process@language#1#2#3{%
4308    \expandafter\addlanguage\csname l@#1\endcsname
4309    \expandafter\language\csname l@#1\endcsname
4310    \edef\languagename{#1}%
4311    \bbl@hook@everylanguage{#1}%
4312    % > luatex
4313    \bbl@get@enc#1::\@@@
4314    \begingroup
4315        \lefthyphenmin\m@ne
4316        \bbl@hook@loadpatterns{#2}%
4317        % > luatex
4318        \ifnum\lefthyphenmin=\m@ne
4319        \else
4320            \expandafter\xdef\csname #1hyphenmins\endcsname{%
4321                \the\lefthyphenmin\the\righthyphenmin}%
4322        \fi
4323    \endgroup
4324    \def\bbl@tempa{#3}%
4325    \ifx\bbl@tempa\@empty\else
4326        \bbl@hook@loadexceptions{#3}%
4327        % > luatex
4328    \fi
4329    \let\bbl@elt\relax
4330    \edef\bbl@languages{%
4331        \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
4332    \ifnum\the\language=\z@
4333        \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4334            \set@hyphenmins\tw@\thr@@\relax
4335        \else
4336            \expandafter\expandafter\expandafter\set@hyphenmins
4337                \csname #1hyphenmins\endcsname
4338        \fi
4339        \the\toks@
```

```
4340    \toks@{}%
4341  \fi}
```

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

```
4342 \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.

```
4343 \def\bbl@hook@everylanguage#1{}
4344 \def\bbl@hook@loadpatterns#1{\input #1\relax}
4345 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
4346 \def\bbl@hook@loadkernel#1{%
4347   \def\addlanguage{\csname newlanguage\endcsname}%
4348   \def\adddialect##1##2{%
4349     \global\chardef##1##2\relax
4350     \wlog{\string##1 = a dialect from \string\language##2}}%
4351   \def\iflanguage##1{%
4352     \expandafter\ifx\csname l@##1\endcsname\relax
4353       \@nolanerr{##1}%
4354     \else
4355       \ifnum\csname l@##1\endcsname=\language
4356         \expandafter\expandafter\expandafter\@firstoftwo
4357       \else
4358         \expandafter\expandafter\expandafter\@secondoftwo
4359       \fi
4360     \fi}%
4361   \def\providehyphenmins##1##2{%
4362     \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
4363       \@namedef{##1hyphenmins}{##2}%
4364     \fi}%
4365   \def\set@hyphenmins##1##2{%
4366     \lefthyphenmin##1\relax
4367     \righthyphenmin##2\relax}%
4368   \def\selectlanguage{%
4369     \errhelp{Selecting a language requires a package supporting it}%
4370     \errmessage{Not loaded}}%
4371   \let\foreignlanguage\selectlanguage
4372   \let\otherlanguage\selectlanguage
4373   \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4374   \def\bbl@usehooks##1##2{}% TODO. Temporary!!
4375   \def\setlocale{%
4376     \errhelp{Find an armchair, sit down and wait}%
4377     \errmessage{Not yet available}}%
4378   \let\uselocale\setlocale
4379   \let\locale\setlocale
4380   \let\selectlocale\setlocale
4381   \let\localename\setlocale
4382   \let\textlocale\setlocale
4383   \let\textlanguage\setlocale
4384   \let\languagetext\setlocale}
4385 \begingroup
4386   \def\AddBabelHook#1#2{%
4387     \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
4388       \def\next{\toks1}%
4389     \else
4390       \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname####1}%
4391     \fi
4392     \next}
4393   \ifx\directlua\@undefined
4394     \ifx\XeTeXinputencoding\@undefined\else
4395       \input xebabel.def
```

153

```
4396       \fi
4397    \else
4398       \input luababel.def
4399    \fi
4400    \openin1 = babel-\bbl@format.cfg
4401    \ifeof1
4402    \else
4403       \input babel-\bbl@format.cfg\relax
4404    \fi
4405    \closein1
4406 \endgroup
4407 \bbl@hook@loadkernel{switch.def}
```

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

```
4408 \openin1 = language.dat
```

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

```
4409 \def\languagename{english}%
4410 \ifeof1
4411   \message{I couldn't find the file language.dat,\space
4412           I will try the file hyphen.tex}
4413   \input hyphen.tex\relax
4414   \chardef\l@english\z@
4415 \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.

```
4416   \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.

```
4417   \loop
4418     \endlinechar\m@ne
4419     \read1 to \bbl@line
4420     \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.

```
4421     \if T\ifeof1F\fi T\relax
4422       \ifx\bbl@line\@empty\else
4423         \edef\bbl@line{\bbl@line\space\space\space}%
4424         \expandafter\process@line\bbl@line\relax
4425       \fi
4426   \repeat
```

Check for the end of the file. We must reverse the test for `\ifeof` without `\else`. Then reactivate the default patterns, and close the configuration file.

```
4427   \begingroup
4428     \def\bbl@elt#1#2#3#4{%
4429       \global\language=#2\relax
4430       \gdef\languagename{#1}%
4431       \def\bbl@elt##1##2##3##4{}}%
4432     \bbl@languages
4433   \endgroup
4434 \fi
4435 \closein1
```

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

```
4436 \if/\the\toks@/\else
4437   \errhelp{language.dat loads no language, only synonyms}
4438   \errmessage{Orphan language synonym}
4439 \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.

```
4440 \let\bbl@line\@undefined
4441 \let\process@line\@undefined
4442 \let\process@synonym\@undefined
4443 \let\process@language\@undefined
4444 \let\bbl@get@enc\@undefined
4445 \let\bbl@hyph@enc\@undefined
4446 \let\bbl@tempa\@undefined
4447 \let\bbl@hook@loadkernel\@undefined
4448 \let\bbl@hook@everylanguage\@undefined
4449 \let\bbl@hook@loadpatterns\@undefined
4450 \let\bbl@hook@loadexceptions\@undefined
4451 ⟨/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].

```
4452 ⟨⟨*More package options⟩⟩ ≡
4453 \chardef\bbl@bidimode\z@
4454 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4455 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
4456 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4457 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4458 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4459 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4460 ⟨⟨/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.

```
4461 ⟨⟨*Font selection⟩⟩ ≡
4462 \bbl@trace{Font handling with fontspec}
4463 \ifx\ExplSyntaxOn\@undefined\else
4464   \def\bbl@fs@warn@nx#1#2{% \bbl@tempfs is the original macro
4465     \in@{,#1,}{,no-script,language-not-exist,}%
4466     \ifin@\else\bbl@tempfs@nx{#1}{#2}\fi}
4467   \def\bbl@fs@warn@nxx#1#2#3{%
4468     \in@{,#1,}{,no-script,language-not-exist,}%
4469     \ifin@\else\bbl@tempfs@nxx{#1}{#2}{#3}\fi}
4470   \def\bbl@loadfontspec{%
4471     \let\bbl@loadfontspec\relax
4472     \ifx\fontspec\@undefined
4473       \usepackage{fontspec}%
4474     \fi}%
4475 \fi
4476 \@onlypreamble\babelfont
4477 \newcommand\babelfont[2][]{%  1=langs/scripts 2=fam
4478   \bbl@foreach{#1}{%
```

```
4479      \expandafter\ifx\csname date##1\endcsname\relax
4480        \IfFileExists{babel-##1.tex}%
4481          {\babelprovide{##1}}%
4482          {}%
4483      \fi}%
4484    \edef\bbl@tempa{#1}%
4485    \def\bbl@tempb{#2}%  Used by \bbl@bblfont
4486    \bbl@loadfontspec
4487    \EnableBabelHook{babel-fontspec}% Just calls \bbl@switchfont
4488    \bbl@bblfont}
4489 \newcommand\bbl@bblfont[2][]{%  1=features 2=fontname, @font=rm|sf|tt
4490    \bbl@ifunset{\bbl@tempb family}%
4491      {\bbl@providefam{\bbl@tempb}}%
4492      {}%
4493    % For the default font, just in case:
4494    \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
4495    \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
4496      {\bbl@csarg\edef{\bbl@tempb dflt@}{<>{#1}{#2}}% save bbl@rmdflt@
4497       \bbl@exp{%
4498         \let\<\bbl@\bbl@tempb dflt@\languagename>\<\bbl@\bbl@tempb dflt@>%
4499         \\\bbl@font@set\<\bbl@\bbl@tempb dflt@\languagename>%
4500                       \<\bbl@tempb default>\<\bbl@tempb family>}}%
4501      {\bbl@foreach\bbl@tempa{%  ie bbl@rmdflt@lang / *scrt
4502         \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:

```
4503 \def\bbl@providefam#1{%
4504    \bbl@exp{%
4505      \\\newcommand\<#1default>{}% Just define it
4506      \\\bbl@add@list\\\bbl@font@fams{#1}%
4507      \\\DeclareRobustCommand\<#1family>{%
4508        \\\not@math@alphabet\<#1family>\relax
4509        % \\\prepare@family@series@update{#1}\<#1default>% TODO. Fails
4510        \\\fontfamily\<#1default>%
4511        \<ifx>\\\UseHooks\\\@undefined\<else>\\\UseHook{#1family}\<fi>%
4512        \\\selectfont}%
4513      \\\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.

```
4514 \def\bbl@nostdfont#1{%
4515    \bbl@ifunset{bbl@WFF@\f@family}%
4516      {\bbl@csarg\gdef{WFF@\f@family}{}%  Flag, to avoid dupl warns
4517       \bbl@infowarn{The current font is not a babel standard family:\\%
4518         #1%
4519         \fontname\font\\%
4520         There is nothing intrinsically wrong with this warning, and\\%
4521         you can ignore it altogether if you do not need these\\%
4522         families. But if they are used in the document, you should be\\%
4523         aware 'babel' will not set Script and Language for them, so\\%
4524         you may consider defining a new family with \string\babelfont.\\%
4525         See the manual for further details about \string\babelfont.\\%
4526         Reported}}
4527      {}}%
4528 \gdef\bbl@switchfont{%
4529    \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
4530    \bbl@exp{%  eg Arabic -> arabic
4531      \lowercase{\edef\\\bbl@tempa{\bbl@cl{sname}}}}%
4532    \bbl@foreach\bbl@font@fams{%
4533      \bbl@ifunset{bbl@##1dflt@\languagename}%      (1) language?
4534        {\bbl@ifunset{bbl@##1dflt@*\bbl@tempa}%      (2) from script?
4535          {\bbl@ifunset{bbl@##1dflt@}%               2=F - (3) from generic?
4536            {}%                                      123=F - nothing!
4537            {\bbl@exp{%                              3=T - from generic
```

```
4538            \global\let\<bbl@##1dflt@\languagename>%
4539                            \<bbl@##1dflt@>}}}%
4540        {\bbl@exp{%                          2=T - from script
4541            \global\let\<bbl@##1dflt@\languagename>%
4542                            \<bbl@##1dflt@*\bbl@tempa>}}}%
4543      {}}%                                   1=T - language, already defined
4544  \def\bbl@tempa{\bbl@nostdfont{}}%  TODO. Don't use \bbl@tempa
4545  \bbl@foreach\bbl@font@fams{%      don't gather with prev for
4546    \bbl@ifunset{bbl@##1dflt@\languagename}%
4547      {\bbl@cs{famrst@##1}%
4548       \global\bbl@csarg\let{famrst@##1}\relax}%
4549      {\bbl@exp{% order is relevant. TODO: but sometimes wrong!
4550          \\\bbl@add\\\originalTeX{%
4551            \\\bbl@font@rst{\bbl@cl{##1dflt}}%
4552                        \<##1default>\<##1family>{##1}}%
4553          \\\bbl@font@set\<bbl@##1dflt@\languagename>% the main part!
4554                      \<##1default>\<##1family>}}}%
4555    \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.

```
4556 \ifx\f@family\@undefined\else   % if latex
4557   \ifcase\bbl@engine            % if pdftex
4558     \let\bbl@ckeckstdfonts\relax
4559   \else
4560     \def\bbl@ckeckstdfonts{%
4561       \begingroup
4562         \global\let\bbl@ckeckstdfonts\relax
4563         \let\bbl@tempa\@empty
4564         \bbl@foreach\bbl@font@fams{%
4565           \bbl@ifunset{bbl@##1dflt@}%
4566             {\@nameuse{##1family}%
4567              \bbl@csarg\gdef{WFF@\f@family}{}% Flag
4568              \bbl@exp{\\\bbl@add\\\bbl@tempa{* \<##1family>= \f@family\\\\%
4569                  \space\space\fontname\font\\\\}}%
4570              \bbl@csarg\xdef{##1dflt@}{\f@family}%
4571              \expandafter\xdef\csname ##1default\endcsname{\f@family}}%
4572             {}}%
4573         \ifx\bbl@tempa\@empty\else
4574           \bbl@infowarn{The following font families will use the default\\%
4575             settings for all or some languages:\\%
4576             \bbl@tempa
4577             There is nothing intrinsically wrong with it, but\\%
4578             'babel' will no set Script and Language, which could\\%
4579              be relevant in some languages. If your document uses\\%
4580              these families, consider redefining them with \string\babelfont.\\%
4581             Reported}%
4582         \fi
4583       \endgroup}
4584   \fi
4585 \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.

```
4586 \def\bbl@font@set#1#2#3{% eg \bbl@rmdflt@lang \rmdefault \rmfamily
4587   \bbl@xin@{<>}{#1}%
4588   \ifin@
4589     \bbl@exp{\\\bbl@fontspec@set\\#1\expandafter\@gobbletwo#1\\#3}%
4590   \fi
4591   \bbl@exp{%              'Unprotected' macros return prev values
4592     \def\\#2{#1}%         eg, \rmdefault{\bbl@rmdflt@lang}
4593     \\\bbl@ifsamestring{#2}{\f@family}%
```

157

```
4594        {\\\#3%
4595          \\\bbl@ifsamestring{\f@series}{\bfdefault}{\\\bfseries}{}%
4596          \let\\\bbl@tempa\relax}%
4597        {}}}
4598 %      TODO - next should be global?, but even local does its job. I'm
4599 %      still not sure -- must investigate:
4600 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
4601   \let\bbl@tempe\bbl@mapselect
4602   \let\bbl@mapselect\relax
4603   \let\bbl@temp@fam#4%         eg, '\rmfamily', to be restored below
4604   \let#4\@empty        %       Make sure \renewfontfamily is valid
4605   \bbl@exp{%
4606     \let\\\bbl@temp@pfam\<\bbl@stripslash#4\space>% eg, '\rmfamily '
4607     \<keys_if_exist:nnF>{fontspec-opentype}{Script/\bbl@cl{sname}}%
4608       {\\\newfontscript{\bbl@cl{sname}}{\bbl@cl{sotf}}}%
4609     \<keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@cl{lname}}%
4610       {\\\newfontlanguage{\bbl@cl{lname}}{\bbl@cl{lotf}}}%
4611     \let\\\bbl@tempfs@nx\<__fontspec_warning:nx>%
4612     \let\<__fontspec_warning:nx>\\\bbl@fs@warn@nx
4613     \let\\\bbl@tempfs@nxx\<__fontspec_warning:nxx>%
4614     \let\<__fontspec_warning:nxx>\\\bbl@fs@warn@nxx
4615     \\\renewfontfamily\\\#4%
4616       [\bbl@cl{lsys},#2]}{#3}% ie \bbl@exp{..}{#3}
4617   \bbl@exp{%
4618     \let\<__fontspec_warning:nx>\\\bbl@tempfs@nx
4619     \let\<__fontspec_warning:nxx>\\\bbl@tempfs@nxx}%
4620   \begingroup
4621     #4%
4622     \xdef#1{\f@family}%     eg, \bbl@rmdflt@lang{FreeSerif(0)}
4623   \endgroup
4624   \let#4\bbl@temp@fam
4625   \bbl@exp{\let\<\bbl@stripslash#4\space>}\bbl@temp@pfam
4626   \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.

```
4627 \def\bbl@font@rst#1#2#3#4{%
4628   \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.

```
4629 \def\bbl@font@fams{rm,sf,tt}
4630 ⟨⟨/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.

```
4631 ⟨⟨*Footnote changes⟩⟩ ≡
4632 \bbl@trace{Bidi footnotes}
4633 \ifnum\bbl@bidimode>\z@
4634   \def\bbl@footnote#1#2#3{%
4635     \@ifnextchar[%
4636       {\bbl@footnote@o{#1}{#2}{#3}}%
4637       {\bbl@footnote@x{#1}{#2}{#3}}}
4638   \long\def\bbl@footnote@x#1#2#3#4{%
4639     \bgroup
4640       \select@language@x{\bbl@main@language}%
4641       \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
4642     \egroup}
4643   \long\def\bbl@footnote@o#1#2#3[#4]#5{%
```

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

Now, the code.

```
4678 ⟨∗xetex⟩
4679 \def\BabelStringsDefault{unicode}
4680 \let\xebbl@stop\relax
4681 \AddBabelHook{xetex}{encodedcommands}{%
4682   \def\bbl@tempa{#1}%
4683   \ifx\bbl@tempa\@empty
4684     \XeTeXinputencoding"bytes"%
4685   \else
4686     \XeTeXinputencoding"#1"%
4687   \fi
4688   \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4689 \AddBabelHook{xetex}{stopcommands}{%
4690   \xebbl@stop
4691   \let\xebbl@stop\relax}
4692 \def\bbl@intraspace#1 #2 #3\@@{%
4693   \bbl@csarg\gdef{xeisp@\languagename}%
4694     {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4695 \def\bbl@intrapenalty#1\@@{%
4696   \bbl@csarg\gdef{xeipn@\languagename}%
4697     {\XeTeXlinebreakpenalty #1\relax}}
4698 \def\bbl@provide@intraspace{%
4699   \bbl@xin@{/s}{/\bbl@cl{lnbrk}}%
4700   \ifin@\else\bbl@xin@{/c}{/\bbl@cl{lnbrk}}\fi
4701   \ifin@
4702     \bbl@ifunset{bbl@intsp@\languagename}{}%
4703       {\expandafter\ifx\csname bbl@intsp@\languagename\endcsname\@empty\else
4704         \ifx\bbl@KVP@intraspace\@nnil
```

```
4705           \bbl@exp{%
4706               \\\bbl@intraspace\bbl@cl{intsp}\\\@@}%
4707         \fi
4708         \ifx\bbl@KVP@intrapenalty\@nnil
4709           \bbl@intrapenalty0\@@
4710         \fi
4711       \fi
4712       \ifx\bbl@KVP@intraspace\@nnil\else % We may override the ini
4713         \expandafter\bbl@intraspace\bbl@KVP@intraspace\@@
4714       \fi
4715       \ifx\bbl@KVP@intrapenalty\@nnil\else
4716         \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
4717       \fi
4718       \bbl@exp{%
4719         % TODO. Execute only once (but redundant):
4720         \\\bbl@add\<extras\languagename>{%
4721           \XeTeXlinebreaklocale "\bbl@cl{tbcp}"%
4722           \<bbl@xeisp@\languagename>%
4723           \<bbl@xeipn@\languagename>}%
4724         \\\bbl@toglobal\<extras\languagename>%
4725         \\\bbl@add\<noextras\languagename>{%
4726           \XeTeXlinebreaklocale ""}%
4727         \\\bbl@toglobal\<noextras\languagename>}%
4728       \ifx\bbl@ispacesize\@undefined
4729         \gdef\bbl@ispacesize{\bbl@cl{xeisp}}%
4730         \ifx\AtBeginDocument\@notprerr
4731           \expandafter\@secondoftwo  % to execute right now
4732         \fi
4733         \AtBeginDocument{\bbl@patchfont{\bbl@ispacesize}}%
4734       \fi}%
4735   \fi}
4736 \ifx\DisableBabelHook\@undefined\endinput\fi
4737 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4738 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
4739 \DisableBabelHook{babel-fontspec}
4740 ⟨⟨Font selection⟩⟩
4741 \def\bbl@provide@extra#1{}
4742 ⟨/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.

```
4743 ⟨∗xetex | texxet⟩
4744 \providecommand\bbl@provide@intraspace{}
4745 \bbl@trace{Redefinitions for bidi layout}
4746 \def\bbl@sspre@caption{%
4747   \bbl@exp{\everyhbox{\\\bbl@textdir\bbl@cs{wdir@\bbl@main@language}}}}
4748 \ifx\bbl@opt@layout\@nnil\else % if layout=..
4749 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
4750 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
4751 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
4752   \def\@hangfrom#1{%
4753     \setbox\@tempboxa\hbox{{#1}}%
4754     \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
4755     \noindent\box\@tempboxa}
4756   \def\raggedright{%
4757     \let\\\@centercr
4758     \bbl@startskip\z@skip
```

```
4759      \@rightskip\@flushglue
4760      \bbl@endskip\@rightskip
4761      \parindent\z@
4762      \parfillskip\bbl@startskip}
4763    \def\raggedleft{%
4764      \let\\\@centercr
4765      \bbl@startskip\@flushglue
4766      \bbl@endskip\z@skip
4767      \parindent\z@
4768      \parfillskip\bbl@endskip}
4769 \fi
4770 \IfBabelLayout{lists}
4771    {\bbl@sreplace\list
4772      {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
4773     \def\bbl@listleftmargin{%
4774      \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
4775     \ifcase\bbl@engine
4776      \def\labelenumii()\theenumii(}% pdftex doesn't reverse ()
4777      \def\p@enumiii{\p@enumii)\theenumii(}%
4778     \fi
4779     \bbl@sreplace\@verbatim
4780      {\leftskip\@totalleftmargin}%
4781      {\bbl@startskip\textwidth
4782       \advance\bbl@startskip-\linewidth}%
4783     \bbl@sreplace\@verbatim
4784      {\rightskip\z@skip}%
4785      {\bbl@endskip\z@skip}}%
4786    {}
4787 \IfBabelLayout{contents}
4788    {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
4789     \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
4790    {}
4791 \IfBabelLayout{columns}
4792    {\bbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbl@outputhbox}%
4793     \def\bbl@outputhbox#1{%
4794      \hb@xt@\textwidth{%
4795       \hskip\columnwidth
4796       \hfil
4797       {\normalcolor\vrule \@width\columnseprule}%
4798       \hfil
4799       \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
4800       \hskip-\textwidth
4801       \hb@xt@\columnwidth{\box\@outputbox \hss}%
4802       \hskip\columnsep
4803       \hskip\columnwidth}}}%
4804    {}
4805 ⟨⟨Footnote changes⟩⟩
4806 \IfBabelLayout{footnotes}%
4807    {\BabelFootnote\footnote\languagename{}{}%
4808     \BabelFootnote\localfootnote\languagename{}{}%
4809     \BabelFootnote\mainfootnote{}{}{}}
4810    {}
```

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.

```
4811 \IfBabelLayout{counters*}%
4812    {\bbl@add\bbl@opt@layout{.counters.}%
4813     \AddToHook{shipout/before}{%
4814      \let\bbl@tempa\babelsublr
4815      \let\babelsublr\@firstofone
4816      \let\bbl@save@thepage\thepage
4817      \protected@edef\thepage{\thepage}%
4818      \let\babelsublr\bbl@tempa}%
```

```
4819    \AddToHook{shipout/after}{%
4820      \let\thepage\bbl@save@thepage}}{}
4821 \IfBabelLayout{counters}%
4822   {\let\bbl@latinarabic=\@arabic
4823    \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
4824    \let\bbl@asciiroman=\@roman
4825    \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciiroman#1}}}%
4826    \let\bbl@asciiRoman=\@Roman
4827    \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}}{}
4828 \fi % end if layout
4829 ⟨/xetex | texxet⟩
```

12.3   8-bit TeX

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

```
4830 ⟨∗texxet⟩
4831 \def\bbl@provide@extra#1{%
4832   % == auto-select encoding ==
4833   \ifx\bbl@encoding@select@off\@empty\else
4834     \bbl@ifunset{bbl@encoding@#1}%
4835       {\def\@elt##1{,##1,}%
4836        \edef\bbl@tempe{\expandafter\@gobbletwo\@fontenc@load@list}%
4837        \count@\z@
4838        \bbl@foreach\bbl@tempe{%
4839          \def\bbl@tempd{##1}%  Save last declared
4840          \advance\count@\@ne}%
4841        \ifnum\count@>\@ne
4842          \getlocaleproperty*\bbl@tempa{#1}{identification/encodings}%
4843          \ifx\bbl@tempa\relax \let\bbl@tempa\@empty \fi
4844          \bbl@replace\bbl@tempa{ }{,}%
4845          \global\bbl@csarg\let{encoding@#1}\@empty
4846          \bbl@xin@{,\bbl@tempd,}{,\bbl@tempa,}%
4847          \ifin@\else % if main encoding included in ini, do nothing
4848            \let\bbl@tempb\relax
4849            \bbl@foreach\bbl@tempa{%
4850              \ifx\bbl@tempb\relax
4851                \bbl@xin@{,##1,}{,\bbl@tempe,}%
4852                \ifin@\def\bbl@tempb{##1}\fi
4853              \fi}%
4854            \ifx\bbl@tempb\relax\else
4855              \bbl@exp{%
4856                \global\<bbl@add>\<bbl@preextras@#1>{\<bbl@encoding@#1>}%
4857              \gdef\<bbl@encoding@#1>{%
4858                \\\babel@save\\\f@encoding
4859                \\\bbl@add\\\originalTeX{\\\selectfont}%
4860                \\\fontencoding{\bbl@tempb}%
4861                \\\selectfont}}%
4862            \fi
4863          \fi
4864        \fi}%
4865      {}%
4866   \fi}
4867 ⟨/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).

```
4868 ⟨∗luatex⟩
4869 \ifx\AddBabelHook\@undefined % When plain.def, babel.sty starts
4870 \bbl@trace{Read language.dat}
4871 \ifx\bbl@readstream\@undefined
4872   \csname newread\endcsname\bbl@readstream
4873 \fi
4874 \begingroup
4875   \toks@{}
4876   \count@\z@ % 0=start, 1=0th, 2=normal
4877   \def\bbl@process@line#1#2 #3 #4 {%
4878     \ifx=#1%
4879       \bbl@process@synonym{#2}%
4880     \else
4881       \bbl@process@language{#1#2}{#3}{#4}%
4882     \fi
4883     \ignorespaces}
4884   \def\bbl@manylang{%
4885     \ifnum\bbl@last>\@ne
4886       \bbl@info{Non-standard hyphenation setup}%
4887     \fi
4888     \let\bbl@manylang\relax}
4889   \def\bbl@process@language#1#2#3{%
4890     \ifcase\count@
4891       \@ifundefined{zth@#1}{\count@\tw@}{\count@\@ne}%
4892     \or
4893       \count@\tw@
4894     \fi
4895     \ifnum\count@=\tw@
4896       \expandafter\addlanguage\csname l@#1\endcsname
4897       \language\allocationnumber
4898       \chardef\bbl@last\allocationnumber
4899       \bbl@manylang
4900       \let\bbl@elt\relax
4901       \xdef\bbl@languages{%
4902         \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
4903     \fi
4904     \the\toks@
4905     \toks@{}}
```

```
4906  \def\bbl@process@synonym@aux#1#2{%
4907    \global\expandafter\chardef\csname l@#1\endcsname#2\relax
4908    \let\bbl@elt\relax
4909    \xdef\bbl@languages{%
4910      \bbl@languages\bbl@elt{#1}{#2}{}{}}}%
4911  \def\bbl@process@synonym#1{%
4912    \ifcase\count@
4913      \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
4914    \or
4915      \@ifundefined{zth@#1}{\bbl@process@synonym@aux{#1}{0}}{}%
4916    \else
4917      \bbl@process@synonym@aux{#1}{\the\bbl@last}%
4918    \fi}
4919  \ifx\bbl@languages\@undefined % Just a (sensible?) guess
4920    \chardef\l@english\z@
4921    \chardef\l@USenglish\z@
4922    \chardef\bbl@last\z@
4923    \global\@namedef{bbl@hyphendata@0}{{hyphen.tex}{}}
4924    \gdef\bbl@languages{%
4925      \bbl@elt{english}{0}{hyphen.tex}{}%
4926      \bbl@elt{USenglish}{0}{}{}}
4927  \else
4928    \global\let\bbl@languages@format\bbl@languages
4929    \def\bbl@elt#1#2#3#4{% Remove all except language 0
4930      \ifnum#2>\z@\else
4931        \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
4932      \fi}%
4933    \xdef\bbl@languages{\bbl@languages}%
4934  \fi
4935  \def\bbl@elt#1#2#3#4{\@namedef{zth@#1}{}} % Define flags
4936  \bbl@languages
4937  \openin\bbl@readstream=language.dat
4938  \ifeof\bbl@readstream
4939    \bbl@warning{I couldn't find language.dat. No additional\\%
4940                patterns loaded. Reported}%
4941  \else
4942    \loop
4943      \endlinechar\m@ne
4944      \read\bbl@readstream to \bbl@line
4945      \endlinechar`\^^M
4946      \if T\ifeof\bbl@readstream F\fi T\relax
4947        \ifx\bbl@line\@empty\else
4948          \edef\bbl@line{\bbl@line\space\space\space}%
4949          \expandafter\bbl@process@line\bbl@line\relax
4950        \fi
4951    \repeat
4952  \fi
4953 \endgroup
4954 \bbl@trace{Macros for reading patterns files}
4955 \def\bbl@get@enc#1:#2:#3\@@@{\def\bbl@hyph@enc{#2}}
4956 \ifx\babelcatcodetablenum\@undefined
4957   \ifx\newcatcodetable\@undefined
4958     \def\babelcatcodetablenum{5211}
4959     \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4960   \else
4961     \newcatcodetable\babelcatcodetablenum
4962     \newcatcodetable\bbl@pattcodes
4963   \fi
4964 \else
4965   \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4966 \fi
4967 \def\bbl@luapatterns#1#2{%
4968   \bbl@get@enc#1::\@@@
```

164

```
4969   \setbox\z@\hbox\bgroup
4970     \begingroup
4971       \savecatcodetable\babelcatcodetablenum\relax
4972       \initcatcodetable\bbl@pattcodes\relax
4973       \catcodetable\bbl@pattcodes\relax
4974         \catcode`\#=6  \catcode`\$=3 \catcode`\&=4 \catcode`\^=7
4975         \catcode`\_=8  \catcode`\{=1 \catcode`\}=2 \catcode`\~=13
4976         \catcode`\@=11 \catcode`\^^I=10 \catcode`\^^J=12
4977         \catcode`\<=12 \catcode`\>=12 \catcode`\*=12 \catcode`\.=12
4978         \catcode`\-=12 \catcode`\/=12 \catcode`\[=12 \catcode`\]=12
4979         \catcode`\`=12 \catcode`\'=12 \catcode`\"=12
4980       \input #1\relax
4981       \catcodetable\babelcatcodetablenum\relax
4982     \endgroup
4983     \def\bbl@tempa{#2}%
4984     \ifx\bbl@tempa\@empty\else
4985       \input #2\relax
4986     \fi
4987   \egroup}%
4988 \def\bbl@patterns@lua#1{%
4989   \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
4990     \csname l@#1\endcsname
4991     \edef\bbl@tempa{#1}%
4992   \else
4993     \csname l@#1:\f@encoding\endcsname
4994     \edef\bbl@tempa{#1:\f@encoding}%
4995   \fi\relax
4996   \@namedef{lu@texhyphen@loaded@\the\language}{}% Temp
4997   \@ifundefined{bbl@hyphendata@\the\language}%
4998     {\def\bbl@elt##1##2##3##4{%
4999        \ifnum##2=\csname l@\bbl@tempa\endcsname % #2=spanish, dutch:OT1...
5000          \def\bbl@tempb{##3}%
5001          \ifx\bbl@tempb\@empty\else % if not a synonymous
5002            \def\bbl@tempc{{##3}{##4}}%
5003          \fi
5004          \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5005        \fi}%
5006      \bbl@languages
5007      \@ifundefined{bbl@hyphendata@\the\language}%
5008        {\bbl@info{No hyphenation patterns were set for\\%
5009                  language '\bbl@tempa'. Reported}}%
5010        {\expandafter\expandafter\expandafter\bbl@luapatterns
5011          \csname bbl@hyphendata@\the\language\endcsname}}{}}
5012 \endinput\fi
5013   % Here ends \ifx\AddBabelHook\@undefined
5014   % A few lines are only read by hyphen.cfg
5015 \ifx\DisableBabelHook\@undefined
5016   \AddBabelHook{luatex}{everylanguage}{%
5017     \def\process@language##1##2##3{%
5018       \def\process@line####1####2 ####3 ####4 {}}}
5019   \AddBabelHook{luatex}{loadpatterns}{%
5020     \input #1\relax
5021     \expandafter\gdef\csname bbl@hyphendata@\the\language\endcsname
5022       {{#1}{}}}
5023   \AddBabelHook{luatex}{loadexceptions}{%
5024     \input #1\relax
5025     \def\bbl@tempb##1##2{{##1}{#1}}%
5026     \expandafter\xdef\csname bbl@hyphendata@\the\language\endcsname
5027       {\expandafter\expandafter\expandafter\bbl@tempb
5028        \csname bbl@hyphendata@\the\language\endcsname}}
5029 \endinput\fi
5030   % Here stops reading code for hyphen.cfg
5031   % The following is read the 2nd time it's loaded
```

```
5032 \begingroup  % TODO - to a lua file
5033 \catcode`\%=12
5034 \catcode`\'=12
5035 \catcode`\"=12
5036 \catcode`\:=12
5037 \directlua{
5038   Babel = Babel or {}
5039   function Babel.bytes(line)
5040     return line:gsub("(.)",
5041       function (chr) return unicode.utf8.char(string.byte(chr)) end)
5042   end
5043   function Babel.begin_process_input()
5044     if luatexbase and luatexbase.add_to_callback then
5045       luatexbase.add_to_callback('process_input_buffer',
5046                                  Babel.bytes,'Babel.bytes')
5047     else
5048       Babel.callback = callback.find('process_input_buffer')
5049       callback.register('process_input_buffer',Babel.bytes)
5050     end
5051   end
5052   function Babel.end_process_input ()
5053     if luatexbase and luatexbase.remove_from_callback then
5054       luatexbase.remove_from_callback('process_input_buffer','Babel.bytes')
5055     else
5056       callback.register('process_input_buffer',Babel.callback)
5057     end
5058   end
5059   function Babel.addpatterns(pp, lg)
5060     local lg = lang.new(lg)
5061     local pats = lang.patterns(lg) or ''
5062     lang.clear_patterns(lg)
5063     for p in pp:gmatch('[^%s]+') do
5064       ss = ''
5065       for i in string.utfcharacters(p:gsub('%d', '')) do
5066         ss = ss .. '%d?' .. i
5067       end
5068       ss = ss:gsub('^%%d%?%.', '%%.') .. '%d?'
5069       ss = ss:gsub('%.%%d%?$', '%%.')
5070       pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
5071       if n == 0 then
5072         tex.sprint(
5073           [[\string\csname\space bbl@info\endcsname{New pattern: ]]
5074           .. p .. [[}]])
5075         pats = pats .. ' ' .. p
5076       else
5077         tex.sprint(
5078           [[\string\csname\space bbl@info\endcsname{Renew pattern: ]]
5079           .. p .. [[}]])
5080       end
5081     end
5082     lang.patterns(lg, pats)
5083   end
5084   Babel.characters = Babel.characters or {}
5085   Babel.ranges = Babel.ranges or {}
5086   function Babel.hlist_has_bidi(head)
5087     local has_bidi = false
5088     local ranges = Babel.ranges
5089     for item in node.traverse(head) do
5090       if item.id == node.id'glyph' then
5091         local itemchar = item.char
5092         local chardata = Babel.characters[itemchar]
5093         local dir = chardata and chardata.d or nil
5094         if not dir then
```

```
5095            for nn, et in ipairs(ranges) do
5096              if itemchar < et[1] then
5097                break
5098              elseif itemchar <= et[2] then
5099                dir = et[3]
5100                break
5101              end
5102            end
5103          end
5104          if dir and (dir == 'al' or dir == 'r') then
5105            has_bidi = true
5106          end
5107        end
5108      end
5109      return has_bidi
5110    end
5111    function Babel.set_chranges_b (script, chrng)
5112      if chrng == '' then return end
5113      texio.write('Replacing ' .. script .. ' script ranges')
5114      Babel.script_blocks[script] = {}
5115      for s, e in string.gmatch(chrng..' ', '(.-)%.%.(.-)%s') do
5116        table.insert(
5117          Babel.script_blocks[script], {tonumber(s,16), tonumber(e,16)})
5118      end
5119    end
5120    function Babel.discard_sublr(str)
5121      if str:find( [[\string\indexentry]] ) and
5122          str:find( [[\string\babelsublr]] ) then
5123       str = str:gsub( [[\string\babelsublr%s*(%b{})]],
5124                    function(m) return m:sub(2,-2) end )
5125      end
5126      return str
5127 end
5128 }
5129 \endgroup
5130 \ifx\newattribute\@undefined\else
5131   \newattribute\bbl@attr@locale
5132   \directlua{ Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale' }
5133   \AddBabelHook{luatex}{beforeextras}{%
5134     \setattribute\bbl@attr@locale\localeid}
5135 \fi
5136 \def\BabelStringsDefault{unicode}
5137 \let\luabbl@stop\relax
5138 \AddBabelHook{luatex}{encodedcommands}{%
5139   \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
5140   \ifx\bbl@tempa\bbl@tempb\else
5141     \directlua{Babel.begin_process_input()}%
5142     \def\luabbl@stop{%
5143       \directlua{Babel.end_process_input()}}%
5144   \fi}%
5145 \AddBabelHook{luatex}{stopcommands}{%
5146   \luabbl@stop
5147   \let\luabbl@stop\relax}
5148 \AddBabelHook{luatex}{patterns}{%
5149  \@ifundefined{bbl@hyphendata@\the\language}%
5150    {\def\bbl@elt##1##2##3##4{%
5151        \ifnum##2=\csname l@##2\endcsname % #2=spanish, dutch:OT1...
5152          \def\bbl@tempb{##3}%
5153          \ifx\bbl@tempb\@empty\else % if not a synonymous
5154            \def\bbl@tempc{{##3}{##4}}%
5155          \fi
5156          \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5157        \fi}%
```

```
5158        \bbl@languages
5159        \@ifundefined{bbl@hyphendata@\the\language}%
5160          {\bbl@info{No hyphenation patterns were set for\\%
5161                    language '#2'. Reported}}%
5162          {\expandafter\expandafter\expandafter\bbl@luapatterns
5163            \csname bbl@hyphendata@\the\language\endcsname}}{}%
5164      \@ifundefined{bbl@patterns@}{}{%
5165        \begingroup
5166          \bbl@xin@{,\number\language,}{,\bbl@pttnlist}%
5167          \ifin@\else
5168            \ifx\bbl@patterns@\@empty\else
5169              \directlua{ Babel.addpatterns(
5170                [[\bbl@patterns@]], \number\language) }%
5171            \fi
5172            \@ifundefined{bbl@patterns@#1}%
5173              \@empty
5174              {\directlua{ Babel.addpatterns(
5175                    [[\space\csname bbl@patterns@#1\endcsname]],
5176                    \number\language) }}%
5177            \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
5178          \fi
5179        \endgroup}%
5180      \bbl@exp{%
5181        \bbl@ifunset{bbl@prehc@\languagename}{}%
5182          {\\\bbl@ifblank{\bbl@cs{prehc@\languagename}}{}%
5183            {\prehyphenchar=\bbl@cl{prehc}\relax}}}}
```

<code>\babelpatterns</code> 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.

```
5184 \@onlypreamble\babelpatterns
5185 \AtEndOfPackage{%
5186   \newcommand\babelpatterns[2][\@empty]{%
5187     \ifx\bbl@patterns@\relax
5188       \let\bbl@patterns@\@empty
5189     \fi
5190     \ifx\bbl@pttnlist\@empty\else
5191       \bbl@warning{%
5192         You must not intermingle \string\selectlanguage\space and\\%
5193         \string\babelpatterns\space or some patterns will not\\%
5194         be taken into account. Reported}%
5195     \fi
5196     \ifx\@empty#1%
5197       \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
5198     \else
5199       \edef\bbl@tempb{\zap@space#1 \@empty}%
5200       \bbl@for\bbl@tempa\bbl@tempb{%
5201         \bbl@fixname\bbl@tempa
5202         \bbl@iflanguage\bbl@tempa{%
5203           \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
5204             \@ifundefined{bbl@patterns@\bbl@tempa}%
5205               \@empty
5206               {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
5207             #2}}}%
5208     \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.

```
5209 % TODO - to a lua file
5210 \directlua{
5211   Babel = Babel or {}
5212   Babel.linebreaking = Babel.linebreaking or {}
5213   Babel.linebreaking.before = {}
5214   Babel.linebreaking.after = {}
5215   Babel.locale = {} % Free to use, indexed by \localeid
5216   function Babel.linebreaking.add_before(func, pos)
5217     tex.print([[\noexpand\csname bbl@luahyphenate\endcsname]])
5218     if pos == nil then
5219       table.insert(Babel.linebreaking.before, func)
5220     else
5221       table.insert(Babel.linebreaking.before, pos, func)
5222     end
5223   end
5224   function Babel.linebreaking.add_after(func)
5225     tex.print([[\noexpand\csname bbl@luahyphenate\endcsname]])
5226     table.insert(Babel.linebreaking.after, func)
5227   end
5228 }
5229 \def\bbl@intraspace#1 #2 #3\@@{%
5230   \directlua{
5231     Babel = Babel or {}
5232     Babel.intraspaces = Babel.intraspaces or {}
5233     Babel.intraspaces['\csname bbl@sbcp@\languagename\endcsname'] = %
5234       {b = #1, p = #2, m = #3}
5235     Babel.locale_props[\the\localeid].intraspace = %
5236       {b = #1, p = #2, m = #3}
5237   }}
5238 \def\bbl@intrapenalty#1\@@{%
5239   \directlua{
5240     Babel = Babel or {}
5241     Babel.intrapenalties = Babel.intrapenalties or {}
5242     Babel.intrapenalties['\csname bbl@sbcp@\languagename\endcsname'] = #1
5243     Babel.locale_props[\the\localeid].intrapenalty = #1
5244   }}
5245 \begingroup
5246 \catcode`\%=12
5247 \catcode`\^=14
5248 \catcode`\'=12
5249 \catcode`\~=12
5250 \gdef\bbl@seaintraspace{^
5251   \let\bbl@seaintraspace\relax
5252   \directlua{
5253     Babel = Babel or {}
5254     Babel.sea_enabled = true
5255     Babel.sea_ranges = Babel.sea_ranges or {}
5256     function Babel.set_chranges (script, chrng)
5257       local c = 0
5258       for s, e in string.gmatch(chrng..' ', '(.-)%.%.(.-)%s') do
5259         Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
5260         c = c + 1
5261       end
5262     end
5263     function Babel.sea_disc_to_space (head)
5264       local sea_ranges = Babel.sea_ranges
5265       local last_char = nil
5266       local quad = 655360        ^% 10 pt = 655360 = 10 * 65536
5267       for item in node.traverse(head) do
5268         local i = item.id
5269         if i == node.id'glyph' then
5270           last_char = item
5271         elseif i == 7 and item.subtype == 3 and last_char
```

169

```
5272              and last_char.char > 0x0C99 then
5273            quad = font.getfont(last_char.font).size
5274            for lg, rg in pairs(sea_ranges) do
5275              if last_char.char > rg[1] and last_char.char < rg[2] then
5276                lg = lg:sub(1, 4)  ^% Remove trailing number of, eg, Cyrl1
5277                local intraspace = Babel.intraspaces[lg]
5278                local intrapenalty = Babel.intrapenalties[lg]
5279                local n
5280                if intrapenalty ~= 0 then
5281                  n = node.new(14, 0)    ^% penalty
5282                  n.penalty = intrapenalty
5283                  node.insert_before(head, item, n)
5284                end
5285                n = node.new(12, 13)     ^% (glue, spaceskip)
5286                node.setglue(n, intraspace.b * quad,
5287                               intraspace.p * quad,
5288                               intraspace.m * quad)
5289                node.insert_before(head, item, n)
5290                node.remove(head, item)
5291              end
5292            end
5293          end
5294        end
5295      end
5296 }^^
5297 \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.

```
5298 \catcode`\%=14
5299 \gdef\bbl@cjkintraspace{%
5300   \let\bbl@cjkintraspace\relax
5301   \directlua{
5302     Babel = Babel or {}
5303     require('babel-data-cjk.lua')
5304     Babel.cjk_enabled = true
5305     function Babel.cjk_linebreak(head)
5306       local GLYPH = node.id'glyph'
5307       local last_char = nil
5308       local quad = 655360      % 10 pt = 655360 = 10 * 65536
5309       local last_class = nil
5310       local last_lang = nil
5311
5312       for item in node.traverse(head) do
5313         if item.id == GLYPH then
5314
5315           local lang = item.lang
5316
5317           local LOCALE = node.get_attribute(item,
5318                   Babel.attr_locale)
5319           local props = Babel.locale_props[LOCALE]
5320
5321           local class = Babel.cjk_class[item.char].c
5322
5323           if props.cjk_quotes and props.cjk_quotes[item.char] then
5324             class = props.cjk_quotes[item.char]
5325           end
```

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

```
5389    }
5390 }
5391 \endgroup
5392 \def\bbl@provide@intraspace{%
5393   \bbl@ifunset{bbl@intsp@\languagename}{}%
5394     {\expandafter\ifx\csname bbl@intsp@\languagename\endcsname\@empty\else
5395       \bbl@xin@{/c}{/\bbl@cl{lnbrk}}%
5396       \ifin@          % cjk
5397         \bbl@cjkintraspace
5398         \directlua{
5399           Babel = Babel or {}
5400           Babel.locale_props = Babel.locale_props or {}
5401           Babel.locale_props[\the\localeid].linebreak = 'c'
5402         }%
5403         \bbl@exp{\\\bbl@intraspace\bbl@cl{intsp}\\\@@}%
5404         \ifx\bbl@KVP@intrapenalty\@nnil
5405           \bbl@intrapenalty0\@@
5406         \fi
5407       \else          % sea
5408         \bbl@seaintraspace
5409         \bbl@exp{\\\bbl@intraspace\bbl@cl{intsp}\\\@@}%
5410         \directlua{
5411           Babel = Babel or {}
5412           Babel.sea_ranges = Babel.sea_ranges or {}
5413           Babel.set_chranges('\bbl@cl{sbcp}',
5414                              '\bbl@cl{chrng}')
5415         }%
5416         \ifx\bbl@KVP@intrapenalty\@nnil
5417           \bbl@intrapenalty0\@@
5418         \fi
5419       \fi
5420     \fi
5421     \ifx\bbl@KVP@intrapenalty\@nnil\else
5422       \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
5423     \fi}}
```

12.7 Arabic justification

```
5424 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5425 \def\bblar@chars{%
5426   0628,0629,062A,062B,062C,062D,062E,062F,0630,0631,0632,0633,%
5427   0634,0635,0636,0637,0638,0639,063A,063B,063C,063D,063E,063F,%
5428   0640,0641,0642,0643,0644,0645,0646,0647,0649}
5429 \def\bblar@elongated{%
5430   0626,0628,062A,062B,0633,0634,0635,0636,063B,%
5431   063C,063D,063E,063F,0641,0642,0643,0644,0646,%
5432   0649,064A}
5433 \begingroup
5434   \catcode`\_=11 \catcode`\:=11
5435   \gdef\bblar@nofswarn{\gdef\msg_warning:nnx##1##2##3{}}
5436 \endgroup
5437 \gdef\bbl@arabicjust{%
5438   \let\bbl@arabicjust\relax
5439   \newattribute\bblar@kashida
5440   \directlua{ Babel.attr_kashida = luatexbase.registernumber'bblar@kashida' }%
5441   \bblar@kashida=\z@
5442   \bbl@patchfont{{\bbl@parsejalt}}%
5443   \directlua{
5444     Babel.arabic.elong_map   = Babel.arabic.elong_map or {}
5445     Babel.arabic.elong_map[\the\localeid]   = {}
5446     luatexbase.add_to_callback('post_linebreak_filter',
5447       Babel.arabic.justify, 'Babel.arabic.justify')
5448     luatexbase.add_to_callback('hpack_filter',
```

```
5449          Babel.arabic.justify_hbox, 'Babel.arabic.justify_hbox')
5450    }}%
5451 % Save both node lists to make replacement. TODO. Save also widths to
5452 % make computations
5453 \def\bblar@fetchjalt#1#2#3#4{%
5454    \bbl@exp{\\\bbl@foreach{#1}}{%
5455      \bbl@ifunset{bblar@JE@##1}%
5456        {\setbox\z@\hbox{^^^^200d\char"##1#2}}%
5457        {\setbox\z@\hbox{^^^^200d\char"\@nameuse{bblar@JE@##1}#2}}%
5458      \directlua{%
5459        local last = nil
5460        for item in node.traverse(tex.box[0].head) do
5461          if item.id == node.id'glyph' and item.char > 0x600 and
5462            not (item.char == 0x200D) then
5463            last = item
5464          end
5465        end
5466        Babel.arabic.#3['##1#4'] = last.char
5467      }}}
5468 % Brute force. No rules at all, yet. The ideal: look at jalt table. And
5469 % perhaps other tables (falt?, cswh?). What about kaf? And diacritic
5470 % positioning?
5471 \gdef\bbl@parsejalt{%
5472    \ifx\addfontfeature\@undefined\else
5473      \bbl@xin@{/e}{/\bbl@cl{lnbrk}}%
5474      \ifin@
5475        \directlua{%
5476          if Babel.arabic.elong_map[\the\localeid][\fontid\font] == nil then
5477            Babel.arabic.elong_map[\the\localeid][\fontid\font] = {}
5478            tex.print([[\string\csname\space bbl@parsejalti\endcsname]])
5479          end
5480        }%
5481      \fi
5482    \fi}
5483 \gdef\bbl@parsejalti{%
5484    \begingroup
5485      \let\bbl@parsejalt\relax      % To avoid infinite loop
5486      \edef\bbl@tempb{\fontid\font}%
5487      \bblar@nofswarn
5488      \bblar@fetchjalt\bblar@elongated{}{from}{}%
5489      \bblar@fetchjalt\bblar@chars{^^^^064a}{from}{a}% Alef maksura
5490      \bblar@fetchjalt\bblar@chars{^^^^0649}{from}{y}% Yeh
5491      \addfontfeature{RawFeature=+jalt}%
5492 % \@namedef{bblar@JE@0643}{06AA}% todo: catch medial kaf
5493      \bblar@fetchjalt\bblar@elongated{}{dest}{}%
5494      \bblar@fetchjalt\bblar@chars{^^^^064a}{dest}{a}%
5495      \bblar@fetchjalt\bblar@chars{^^^^0649}{dest}{y}%
5496      \directlua{%
5497        for k, v in pairs(Babel.arabic.from) do
5498          if Babel.arabic.dest[k] and
5499            not (Babel.arabic.from[k] == Babel.arabic.dest[k]) then
5500            Babel.arabic.elong_map[\the\localeid][\bbl@tempb]
5501              [Babel.arabic.from[k]] = Babel.arabic.dest[k]
5502          end
5503        end
5504      }%
5505    \endgroup}
5506 %
5507 \begingroup
5508 \catcode`#=11
5509 \catcode`~=11
5510 \directlua{
5511
```

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

```
5575            node.set_attribute(n.prev, KASHIDA, 0)
5576          end
5577        end
5578
5579      % Tatwil
5580      if Babel.kashida_wts then
5581        local k_wt = node.get_attribute(n, KASHIDA)
5582        if k_wt > 0 then % todo. parameter for multi inserts
5583          table.insert(k_list, {node = n, weight = k_wt, pos = pos_inline})
5584        end
5585      end
5586
5587    end % of node.traverse_id
5588
5589    if #elongs == 0 and #k_list == 0 then goto next_line end
5590    full  = line.width
5591    shift = line.shift
5592    goal  = full * Babel.arabic.justify_factor % A bit crude
5593    width = node.dimensions(line.head)    % The 'natural' width
5594
5595    % == Elongated ==
5596    % Original idea taken from 'chikenize'
5597    while (#elongs > 0 and width < goal) do
5598      subst_done = true
5599      local x = #elongs
5600      local curr = elongs[x].node
5601      local oldchar = curr.char
5602      curr.char = elong_map[elongs[x].locale][curr.font][curr.char]
5603      width = node.dimensions(line.head)  % Check if the line is too wide
5604      % Substitute back if the line would be too wide and break:
5605      if width > goal then
5606        curr.char = oldchar
5607        break
5608      end
5609      % If continue, pop the just substituted node from the list:
5610      table.remove(elongs, x)
5611    end
5612
5613    % == Tatwil ==
5614    if #k_list == 0 then goto next_line end
5615
5616    width = node.dimensions(line.head)     % The 'natural' width
5617    k_curr = #k_list
5618    wt_pos = 1
5619
5620    while width < goal do
5621      subst_done = true
5622      k_item = k_list[k_curr].node
5623      if k_list[k_curr].weight == Babel.kashida_wts[wt_pos] then
5624        d = node.copy(k_item)
5625        d.char = 0x0640
5626        line.head, new = node.insert_after(line.head, k_item, d)
5627        width_new = node.dimensions(line.head)
5628        if width > goal or width == width_new then
5629          node.remove(line.head, new) % Better compute before
5630          break
5631        end
5632        width = width_new
5633      end
5634      if k_curr == 1 then
5635        k_curr = #k_list
5636        wt_pos = (wt_pos >= table.getn(Babel.kashida_wts)) and 1 or wt_pos+1
5637      else
```

```
5638        k_curr = k_curr - 1
5639      end
5640    end
5641
5642    ::next_line::
5643
5644    % Must take into account marks and ins, see luatex manual.
5645    % Have to be executed only if there are changes. Investigate
5646    % what's going on exactly.
5647    if subst_done and not gc then
5648      d = node.hpack(line.head, full, 'exactly')
5649      d.shift = shift
5650      node.insert_before(head, line, d)
5651      node.remove(head, line)
5652    end
5653  end % if process line
5654 end
5655 }
5656 \endgroup
5657 \fi\fi % Arabic just block
```

12.8   Common stuff

```
5658 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
5659 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
5660 \DisableBabelHook{babel-fontspec}
5661 ⟨⟨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.

```
5662 % TODO - to a lua file
5663 \directlua{
5664 Babel.script_blocks = {
5665  ['dflt'] = {},
5666  ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
5667             {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
5668  ['Armn'] = {{0x0530, 0x058F}},
5669  ['Beng'] = {{0x0980, 0x09FF}},
5670  ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
5671  ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
5672  ['Cyrl'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
5673             {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
5674  ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
5675  ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
5676             {0xAB00, 0xAB2F}},
5677  ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
5678  % Don't follow strictly Unicode, which places some Coptic letters in
5679  % the 'Greek and Coptic' block
5680  ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
5681  ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
5682             {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
5683             {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
5684             {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
5685             {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
5686             {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
5687  ['Hebr'] = {{0x0590, 0x05FF}},
5688  ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
```

```
5689                {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
5690    ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
5691    ['Knda'] = {{0x0C80, 0x0CFF}},
5692    ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
5693                {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
5694                {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
5695    ['Laoo'] = {{0x0E80, 0x0EFF}},
5696    ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
5697                {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
5698                {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
5699    ['Mahj'] = {{0x11150, 0x1117F}},
5700    ['Mlym'] = {{0x0D00, 0x0D7F}},
5701    ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
5702    ['Orya'] = {{0x0B00, 0x0B7F}},
5703    ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
5704    ['Syrc'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
5705    ['Taml'] = {{0x0B80, 0x0BFF}},
5706    ['Telu'] = {{0x0C00, 0x0C7F}},
5707    ['Tfng'] = {{0x2D30, 0x2D7F}},
5708    ['Thai'] = {{0x0E00, 0x0E7F}},
5709    ['Tibt'] = {{0x0F00, 0x0FFF}},
5710    ['Vaii'] = {{0xA500, 0xA63F}},
5711    ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
5712 }
5713
5714 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
5715 Babel.script_blocks.Hant = Babel.script_blocks.Hans
5716 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
5717
5718 function Babel.locale_map(head)
5719   if not Babel.locale_mapped then return head end
5720
5721   local LOCALE = Babel.attr_locale
5722   local GLYPH = node.id('glyph')
5723   local inmath = false
5724   local toloc_save
5725   for item in node.traverse(head) do
5726     local toloc
5727     if not inmath and item.id == GLYPH then
5728       % Optimization: build a table with the chars found
5729       if Babel.chr_to_loc[item.char] then
5730         toloc = Babel.chr_to_loc[item.char]
5731       else
5732         for lc, maps in pairs(Babel.loc_to_scr) do
5733           for _, rg in pairs(maps) do
5734             if item.char >= rg[1] and item.char <= rg[2] then
5735               Babel.chr_to_loc[item.char] = lc
5736               toloc = lc
5737               break
5738             end
5739           end
5740         end
5741       end
5742       % Now, take action, but treat composite chars in a different
5743       % fashion, because they 'inherit' the previous locale. Not yet
5744       % optimized.
5745       if not toloc and
5746           (item.char >= 0x0300 and item.char <= 0x036F) or
5747           (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
5748           (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
5749         toloc = toloc_save
5750       end
5751       if toloc and Babel.locale_props[toloc] and
```

```
5752        Babel.locale_props[toloc].letters and
5753          tex.getcatcode(item.char) \string~= 11 then
5754        toloc = nil
5755      end
5756      if toloc and toloc > -1 then
5757        if Babel.locale_props[toloc].lg then
5758          item.lang = Babel.locale_props[toloc].lg
5759          node.set_attribute(item, LOCALE, toloc)
5760        end
5761        if Babel.locale_props[toloc]['/'..item.font] then
5762          item.font = Babel.locale_props[toloc]['/'..item.font]
5763        end
5764        toloc_save = toloc
5765      end
5766    elseif not inmath and item.id == 7 then % Apply recursively
5767      item.replace = item.replace and Babel.locale_map(item.replace)
5768      item.pre     = item.pre and Babel.locale_map(item.pre)
5769      item.post    = item.post and Babel.locale_map(item.post)
5770    elseif item.id == node.id'math' then
5771      inmath = (item.subtype == 0)
5772    end
5773  end
5774  return head
5775 end
5776 }
```

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

```
5777 \newcommand\babelcharproperty[1]{%
5778   \count@=#1\relax
5779   \ifvmode
5780     \expandafter\bbl@chprop
5781   \else
5782     \bbl@error{\string\babelcharproperty\space can be used only in\\%
5783               vertical mode (preamble or between paragraphs)}%
5784               {See the manual for futher info}%
5785   \fi}
5786 \newcommand\bbl@chprop[3][\the\count@]{%
5787   \@tempcnta=#1\relax
5788   \bbl@ifunset{bbl@chprop@#2}%
5789     {\bbl@error{No property named '#2'. Allowed values are\\%
5790               direction (bc), mirror (bmg), and linebreak (lb)}%
5791               {See the manual for futher info}}%
5792     {}%
5793   \loop
5794     \bbl@cs{chprop@#2}{#3}%
5795   \ifnum\count@<\@tempcnta
5796     \advance\count@\@ne
5797   \repeat}
5798 \def\bbl@chprop@direction#1{%
5799   \directlua{
5800     Babel.characters[\the\count@] =  Babel.characters[\the\count@] or {}
5801     Babel.characters[\the\count@]['d'] = '#1'
5802   }}
5803 \let\bbl@chprop@bc\bbl@chprop@direction
5804 \def\bbl@chprop@mirror#1{%
5805   \directlua{
5806     Babel.characters[\the\count@] =  Babel.characters[\the\count@] or {}
5807     Babel.characters[\the\count@]['m'] = '\number#1'
5808   }}
5809 \let\bbl@chprop@bmg\bbl@chprop@mirror
5810 \def\bbl@chprop@linebreak#1{%
5811   \directlua{
```

178

```
5812     Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
5813     Babel.cjk_characters[\the\count@]['c'] = '#1'
5814   }}
5815 \let\bbl@chprop@lb\bbl@chprop@linebreak
5816 \def\bbl@chprop@locale#1{%
5817   \directlua{
5818     Babel.chr_to_loc = Babel.chr_to_loc or {}
5819     Babel.chr_to_loc[\the\count@] =
5820       \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@@#1}}\space
5821   }}
```

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.

```
5822 \directlua{
5823   Babel.nohyphenation = \the\l@nohyphenation
5824 }
```

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).

```
5825 \begingroup
5826 \catcode`\~=12
5827 \catcode`\%=12
5828 \catcode`\&=14
5829 \catcode`\|=12
5830 \gdef\babelprehyphenation{&%
5831   \@ifnextchar[{\bbl@settransform{0}}{\bbl@settransform{0}[]}}
5832 \gdef\babelposthyphenation{&%
5833   \@ifnextchar[{\bbl@settransform{1}}{\bbl@settransform{1}[]}}
5834 \gdef\bbl@postlinebreak{\bbl@settransform{2}[]} &% WIP
5835 \gdef\bbl@settransform#1[#2]#3#4#5{&%
5836   \ifcase#1
5837     \bbl@activateprehyphen
5838   \or
5839     \bbl@activateposthyphen
5840   \fi
5841   \begingroup
5842     \def\babeltempa{\bbl@add@list\babeltempb}&%
5843     \let\babeltempb\@empty
5844     \def\bbl@tempa{#5}&%
5845     \bbl@replace\bbl@tempa{,}{ ,}&% TODO. Ugly trick to preserve {}
5846     \expandafter\bbl@foreach\expandafter{\bbl@tempa}{&%
5847       \bbl@ifsamestring{##1}{remove}&%
5848         {\bbl@add@list\babeltempb{nil}}&%
5849         {\directlua{
5850           local rep = [=[##1]=]
5851           rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
5852           rep = rep:gsub('^%s*(insert)%s*,', 'insert = true, ')
5853          rep = rep:gsub('(string)%s*=%s*([^%s,]*)', Babel.capture_func)
5854          if #1 == 0 or #1 == 2 then
5855            rep = rep:gsub('(space)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
5856              'space = {' .. '%2, %3, %4' .. '}')
5857            rep = rep:gsub('(spacefactor)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
5858              'spacefactor = {' .. '%2, %3, %4' .. '}')
5859            rep = rep:gsub('(kashida)%s*=%s*([^%s,]*)', Babel.capture_kashida)
5860          else
5861            rep = rep:gsub(    '(no)%s*=%s*([^%s,]*)', Babel.capture_func)
5862            rep = rep:gsub(    '(pre)%s*=%s*([^%s,]*)', Babel.capture_func)
```

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

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

```
5989       {\bbl@csarg\setattribute{ATR@#1@\languagename @}\@ne}}
5990 \DeclareRobustCommand\disablelocaletransform[1]{%
5991   \bbl@ifunset{bbl@ATR@#1@\languagename @}%
5992     {\bbl@error
5993       {'#1' for '\languagename' cannot be disabled.\\%
5994        Maybe there is a typo or it's a font-dependent transform}%
5995       {See the manual for further details.}}%
5996     {\bbl@csarg\unsetattribute{ATR@#1@\languagename @}}}
5997 \def\bbl@activateposthyphen{%
5998   \let\bbl@activateposthyphen\relax
5999   \directlua{
6000     require('babel-transforms.lua')
6001     Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
6002   }}
6003 \def\bbl@activateprehyphen{%
6004   \let\bbl@activateprehyphen\relax
6005   \directlua{
6006     require('babel-transforms.lua')
6007     Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
6008   }}
```

12.10 Bidi

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

```
6009 \def\bbl@activate@preotf{%
6010   \let\bbl@activate@preotf\relax  % only once
6011   \directlua{
6012     Babel = Babel or {}
6013     %
6014     function Babel.pre_otfload_v(head)
6015       if Babel.numbers and Babel.digits_mapped then
6016         head = Babel.numbers(head)
6017       end
6018       if Babel.bidi_enabled then
6019         head = Babel.bidi(head, false, dir)
6020       end
6021       return head
6022     end
6023     %
6024     function Babel.pre_otfload_h(head, gc, sz, pt, dir)
6025       if Babel.numbers and Babel.digits_mapped then
6026         head = Babel.numbers(head)
6027       end
6028       if Babel.bidi_enabled then
6029         head = Babel.bidi(head, false, dir)
6030       end
6031       return head
6032     end
6033     %
6034     luatexbase.add_to_callback('pre_linebreak_filter',
6035       Babel.pre_otfload_v,
6036       'Babel.pre_otfload_v',
6037       luatexbase.priority_in_callback('pre_linebreak_filter',
6038         'luaotfload.node_processor') or nil)
6039     %
6040     luatexbase.add_to_callback('hpack_filter',
6041       Babel.pre_otfload_h,
6042       'Babel.pre_otfload_h',
6043       luatexbase.priority_in_callback('hpack_filter',
6044         'luaotfload.node_processor') or nil)
6045   }}
```

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=.

```
6046 \ifnum\bbl@bidimode>\@ne % Excludes default=1
6047   \let\bbl@beforeforeign\leavevmode
6048   \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
6049   \RequirePackage{luatexbase}
6050   \bbl@activate@preotf
6051   \directlua{
6052     require('babel-data-bidi.lua')
6053     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
6054       require('babel-bidi-basic.lua')
6055     \or
6056       require('babel-bidi-basic-r.lua')
6057     \fi}
6058   \newattribute\bbl@attr@dir
6059   \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
6060   \bbl@exp{\output{\bodydir\pagedir\the\output}}
6061 \fi
6062 \chardef\bbl@thetextdir\z@
6063 \chardef\bbl@thepardir\z@
6064 \def\bbl@getluadir#1{%
6065   \directlua{
6066     if tex.#1dir == 'TLT' then
6067       tex.sprint('0')
6068     elseif tex.#1dir == 'TRT' then
6069       tex.sprint('1')
6070     end}}
6071 \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
6072   \ifcase#3\relax
6073     \ifcase\bbl@getluadir{#1}\relax\else
6074       #2 TLT\relax
6075     \fi
6076   \else
6077     \ifcase\bbl@getluadir{#1}\relax
6078       #2 TRT\relax
6079     \fi
6080   \fi}
6081 % ..OOPPTT, with masks 0xC (par dir) and 0x3 (text dir)
6082 \def\bbl@thedir{0}
6083 \def\bbl@textdir#1{%
6084   \bbl@setluadir{text}\textdir{#1}%
6085   \chardef\bbl@thetextdir#1\relax
6086   \edef\bbl@thedir{\the\numexpr\bbl@thepardir*4+#1}%
6087   \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*4+#1}}
6088 \def\bbl@pardir#1{%  Used twice
6089   \bbl@setluadir{par}\pardir{#1}%
6090   \chardef\bbl@thepardir#1\relax}
6091 \def\bbl@bodydir{\bbl@setluadir{body}\bodydir}%   Used once
6092 \def\bbl@pagedir{\bbl@setluadir{page}\pagedir}%   Unused
6093 \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.

```
6094 \ifnum\bbl@bidimode>\z@
6095   \def\bbl@insidemath{0}%
6096   \def\bbl@everymath{\def\bbl@insidemath{1}}
6097   \def\bbl@everydisplay{\def\bbl@insidemath{2}}
6098   \frozen@everymath\expandafter{%
6099     \expandafter\bbl@everymath\the\frozen@everymath}
6100   \frozen@everydisplay\expandafter{%
6101     \expandafter\bbl@everydisplay\the\frozen@everydisplay}
6102   \AtBeginDocument{
```

```
6103      \directlua{
6104        function Babel.math_box_dir(head)
6105          if not (token.get_macro('bbl@insidemath') == '0') then
6106            if Babel.hlist_has_bidi(head) then
6107              local d = node.new(node.id'dir')
6108              d.dir = '+TRT'
6109              node.insert_before(head, node.has_glyph(head), d)
6110              for item in node.traverse(head) do
6111                node.set_attribute(item,
6112                  Babel.attr_dir, token.get_macro('bbl@thedir'))
6113              end
6114            end
6115          end
6116          return head
6117        end
6118        luatexbase.add_to_callback("hpack_filter", Babel.math_box_dir,
6119          "Babel.math_box_dir", 0)
6120  }}%
6121 \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.

```
6122 \bbl@trace{Redefinitions for bidi layout}
6123 %
6124 ⟨⟨*More package options⟩⟩ ≡
6125 \chardef\bbl@eqnpos\z@
6126 \DeclareOption{leqno}{\chardef\bbl@eqnpos\@ne}
6127 \DeclareOption{fleqn}{\chardef\bbl@eqnpos\tw@}
6128 ⟨⟨/More package options⟩⟩
6129 %
6130 \ifnum\bbl@bidimode>\z@
6131   \ifx\matheqdirmode\@undefined\else
6132     \matheqdirmode\@ne % A luatex primitive
6133   \fi
6134   \let\bbl@eqnodir\relax
6135   \def\bbl@eqdel{()}
6136   \def\bbl@eqnum{%
6137     {\normalfont\normalcolor
6138      \expandafter\@firstoftwo\bbl@eqdel
6139      \theequation
6140      \expandafter\@secondoftwo\bbl@eqdel}}
6141   \def\bbl@puteqno#1{\eqno\hbox{#1}}
6142   \def\bbl@putleqno#1{\leqno\hbox{#1}}
6143   \def\bbl@eqno@flip#1{%
6144     \ifdim\predisplaysize=-\maxdimen
6145       \eqno
6146       \hb@xt@.01pt{\hb@xt@\displaywidth{\hss{#1}}\hss}%
6147     \else
6148       \leqno\hbox{#1}%
6149     \fi}
6150   \def\bbl@leqno@flip#1{%
```

```
6151        \ifdim\predisplaysize=-\maxdimen
6152          \leqno
6153          \hb@xt@.01pt{\hss\hb@xt@\displaywidth{{#1}\hss}}%
6154        \else
6155          \eqno\hbox{#1}%
6156        \fi}
6157    \AtBeginDocument{%
6158      \ifx\bbl@noamsmath\relax\else
6159      \ifx\maketag@@@\@undefined % Normal equation, eqnarray
6160        \AddToHook{env/equation/begin}{%
6161          \ifnum\bbl@thetextdir>\z@
6162            \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6163            \let\@eqnnum\bbl@eqnum
6164            \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6165            \chardef\bbl@thetextdir\z@
6166            \bbl@add\normalfont{\bbl@eqnodir}%
6167            \ifcase\bbl@eqnpos
6168              \let\bbl@puteqno\bbl@eqno@flip
6169            \or
6170              \let\bbl@puteqno\bbl@leqno@flip
6171            \fi
6172          \fi}%
6173        \ifnum\bbl@eqnpos=\tw@\else
6174          \def\endequation{\bbl@puteqno{\@eqnnum}$$\@ignoretrue}%
6175        \fi
6176        \AddToHook{env/eqnarray/begin}{%
6177          \ifnum\bbl@thetextdir>\z@
6178            \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6179            \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6180            \chardef\bbl@thetextdir\z@
6181            \bbl@add\normalfont{\bbl@eqnodir}%
6182            \ifnum\bbl@eqnpos=\@ne
6183              \def\@eqnnum{%
6184                \setbox\z@\hbox{\bbl@eqnum}%
6185                \hbox to0.01pt{\hss\hbox to\displaywidth{\box\z@\hss}}}%
6186            \else
6187              \let\@eqnnum\bbl@eqnum
6188            \fi
6189          \fi}
6190      % Hack. YA luatex bug?:
6191      \expandafter\bbl@sreplace\csname] \endcsname{$$}{\eqno\kern.001pt$$}%
6192      \else % amstex
6193        \bbl@exp{% Hack to hide maybe undefined conditionals:
6194          \chardef\bbl@eqnpos=0%
6195            \<iftagsleft@>1\<else>\<if@fleqn>2\<fi>\<fi>\relax}%
6196        \ifnum\bbl@eqnpos=\@ne
6197          \let\bbl@ams@lap\hbox
6198        \else
6199          \let\bbl@ams@lap\llap
6200        \fi
6201        \ExplSyntaxOn
6202        \bbl@sreplace\intertext@{\normalbaselines}%
6203          {\normalbaselines
6204            \ifx\bbl@eqnodir\relax\else\bbl@pardir\@ne\bbl@eqnodir\fi}%
6205        \ExplSyntaxOff
6206        \def\bbl@ams@tagbox#1#2{#1{\bbl@eqnodir#2}}% #1=hbox|@lap|flip
6207        \ifx\bbl@ams@lap\hbox % leqno
6208          \def\bbl@ams@flip#1{%
6209            \hbox to 0.01pt{\hss\hbox to\displaywidth{{#1}\hss}}}%
6210        \else % eqno
6211          \def\bbl@ams@flip#1{%
6212            \hbox to 0.01pt{\hbox to\displaywidth{\hss{#1}}\hss}}%
6213        \fi
```

185

```
6214        \def\bbl@ams@preset#1{%
6215          \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6216          \ifnum\bbl@thetextdir>\z@
6217            \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6218            \bbl@sreplace\textdef@{\hbox}{\bbl@ams@tagbox\hbox}%
6219            \bbl@sreplace\maketag@@@{\hbox}{\bbl@ams@tagbox#1}%
6220          \fi}%
6221        \ifnum\bbl@eqnpos=\tw@\else
6222          \def\bbl@ams@equation{%
6223            \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6224            \ifnum\bbl@thetextdir>\z@
6225              \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6226              \chardef\bbl@thetextdir\z@
6227              \bbl@add\normalfont{\bbl@eqnodir}%
6228              \ifcase\bbl@eqnpos
6229                \def\veqno##1##2{\bbl@eqno@flip{##1##2}}%
6230              \or
6231                \def\veqno##1##2{\bbl@leqno@flip{##1##2}}%
6232              \fi
6233            \fi}%
6234          \AddToHook{env/equation/begin}{\bbl@ams@equation}%
6235          \AddToHook{env/equation*/begin}{\bbl@ams@equation}%
6236        \fi
6237        \AddToHook{env/cases/begin}{\bbl@ams@preset\bbl@ams@lap}%
6238        \AddToHook{env/multline/begin}{\bbl@ams@preset\hbox}%
6239        \AddToHook{env/gather/begin}{\bbl@ams@preset\bbl@ams@lap}%
6240        \AddToHook{env/gather*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6241        \AddToHook{env/align/begin}{\bbl@ams@preset\bbl@ams@lap}%
6242        \AddToHook{env/align*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6243        \AddToHook{env/eqnalign/begin}{\bbl@ams@preset\hbox}%
6244        % Hackish, for proper alignment. Don't ask me why it works!:
6245        \bbl@exp{% Avoid a 'visible' conditional
6246          \\\AddToHook{env/align*/end}{\<iftag@>\<else>\\\tag*{}\<fi>}}%
6247        \AddToHook{env/flalign/begin}{\bbl@ams@preset\hbox}%
6248        \AddToHook{env/split/before}{%
6249          \def\bbl@mathboxdir{\def\bbl@insidemath{1}}%
6250          \ifnum\bbl@thetextdir>\z@
6251            \bbl@ifsamestring\@currenvir{equation}%
6252              {\ifx\bbl@ams@lap\hbox % leqno
6253                 \def\bbl@ams@flip#1{%
6254                   \hbox to 0.01pt{\hbox to\displaywidth{{#1}\hss}\hss}}%
6255               \else
6256                 \def\bbl@ams@flip#1{%
6257                   \hbox to 0.01pt{\hss\hbox to\displaywidth{\hss{#1}}}}%
6258               \fi}%
6259              {}%
6260          \fi}%
6261    \fi\fi}
6262 \fi
6263 \def\bbl@provide@extra#1{%
6264   % == Counters: mapdigits ==
6265   % Native digits
6266   \ifx\bbl@KVP@mapdigits\@nnil\else
6267     \bbl@ifunset{bbl@dgnat@\languagename}{}%
6268       {\RequirePackage{luatexbase}%
6269        \bbl@activate@preotf
6270        \directlua{
6271          Babel = Babel or {}  %%% -> presets in luababel
6272          Babel.digits_mapped = true
6273          Babel.digits = Babel.digits or {}
6274          Babel.digits[\the\localeid] =
6275            table.pack(string.utfvalue('\bbl@cl{dgnat}'))
6276          if not Babel.numbers then
```

186

```
6277            function Babel.numbers(head)
6278              local LOCALE = Babel.attr_locale
6279              local GLYPH = node.id'glyph'
6280              local inmath = false
6281              for item in node.traverse(head) do
6282                if not inmath and item.id == GLYPH then
6283                  local temp = node.get_attribute(item, LOCALE)
6284                  if Babel.digits[temp] then
6285                    local chr = item.char
6286                    if chr > 47 and chr < 58 then
6287                      item.char = Babel.digits[temp][chr-47]
6288                    end
6289                  end
6290                elseif item.id == node.id'math' then
6291                  inmath = (item.subtype == 0)
6292                end
6293              end
6294              return head
6295            end
6296          end
6297      }}%
6298    \fi
6299  % == transforms ==
6300  \ifx\bbl@KVP@transforms\@nnil\else
6301    \def\bbl@elt##1##2##3{%
6302      \in@{$transforms.}{$##1}%
6303      \ifin@
6304        \def\bbl@tempa{##1}%
6305        \bbl@replace\bbl@tempa{transforms.}{}%
6306        \bbl@carg\bbl@transforms{babel\bbl@tempa}{##2}{##3}%
6307      \fi}%
6308    \csname bbl@inidata@\languagename\endcsname
6309    \bbl@release@transforms\relax % \relax closes the last item.
6310  \fi}
6311 % Start tabular here:
6312 \def\localerestoredirs{%
6313  \ifcase\bbl@thetextdir
6314    \ifnum\textdirection=\z@\else\textdir TLT\fi
6315  \else
6316    \ifnum\textdirection=\@ne\else\textdir TRT\fi
6317  \fi
6318  \ifcase\bbl@thepardir
6319    \ifnum\pardirection=\z@\else\pardir TLT\bodydir TLT\fi
6320  \else
6321    \ifnum\pardirection=\@ne\else\pardir TRT\bodydir TRT\fi
6322  \fi}
6323 \IfBabelLayout{tabular}%
6324  {\chardef\bbl@tabular@mode\tw@}% All RTL
6325  {\IfBabelLayout{notabular}%
6326    {\chardef\bbl@tabular@mode\z@}%
6327    {\chardef\bbl@tabular@mode\@ne}}% Mixed, with LTR cols
6328 \ifnum\bbl@bidimode>\@ne
6329  \ifnum\bbl@tabular@mode=\@ne
6330    \let\bbl@parabefore\relax
6331    \AddToHook{para/before}{\bbl@parabefore}
6332    \AtBeginDocument{%
6333      \bbl@replace\@tabular{$}{$%
6334        \def\bbl@insidemath{0}%
6335        \def\bbl@parabefore{\localerestoredirs}}%
6336      \ifnum\bbl@tabular@mode=\@ne
6337        \bbl@ifunset{@tabclassz}{}{%
6338          \bbl@exp{% Hide conditionals
6339            \\\bbl@sreplace\\\@tabclassz
```

```
6340              {\<ifcase>\\\@chnum}%
6341              {\\\localerestoredirs\<ifcase>\\\@chnum}}}%
6342          \@ifpackageloaded{colortbl}%
6343            {\bbl@sreplace\@classz
6344              {\hbox\bgroup\bgroup}{\hbox\bgroup\bgroup\localerestoredirs}}%
6345            {\@ifpackageloaded{array}%
6346              {\bbl@exp{% Hide conditionals
6347                \\\bbl@sreplace\\\@classz
6348                  {\<ifcase>\\\@chnum}%
6349                  {\bgroup\\\localerestoredirs\<ifcase>\\\@chnum}%
6350                \\\bbl@sreplace\\\@classz
6351                  {\\\do@row@strut\<fi>}{\\\do@row@strut\<fi>\egroup}}}%
6352              {}}%
6353      \fi}
6354    \fi
6355 \fi
6356 \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.

```
6357 \ifnum\bbl@bidimode>\z@
6358   \def\bbl@nextfake#1{%  non-local changes, use always inside a group!
6359     \bbl@exp{%
6360       \def\\\bbl@insidemath{0}%
6361       \mathdir\the\bodydir
6362       #1%                Once entered in math, set boxes to restore values
6363       \<ifmmode>%
6364         \everyvbox{%
6365           \the\everyvbox
6366           \bodydir\the\bodydir
6367           \mathdir\the\mathdir
6368           \everyhbox{\the\everyhbox}%
6369           \everyvbox{\the\everyvbox}}%
6370         \everyhbox{%
6371           \the\everyhbox
6372           \bodydir\the\bodydir
6373           \mathdir\the\mathdir
6374           \everyhbox{\the\everyhbox}%
6375           \everyvbox{\the\everyvbox}}%
6376       \<fi>}}%
6377   \def\@hangfrom#1{%
6378     \setbox\@tempboxa\hbox{{#1}}%
6379     \hangindent\wd\@tempboxa
6380     \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
6381       \shapemode\@ne
6382     \fi
6383     \noindent\box\@tempboxa}
6384 \fi
6385 \IfBabelLayout{tabular}
6386   {\let\bbl@OL@@tabular\@tabular
6387    \bbl@replace\@tabular{$}{\bbl@nextfake$}%
6388    \let\bbl@NL@@tabular\@tabular
6389    \AtBeginDocument{%
6390      \ifx\bbl@NL@@tabular\@tabular\else
6391        \bbl@replace\@tabular{$}{\bbl@nextfake$}%
6392        \let\bbl@NL@@tabular\@tabular
6393      \fi}}
6394    {}
6395 \IfBabelLayout{lists}
6396   {\let\bbl@OL@list\list
6397    \bbl@sreplace\list{\parshape}{\bbl@listparshape}%
```

```
6398    \let\bbl@NL@list\list
6399    \def\bbl@listparshape#1#2#3{%
6400      \parshape #1 #2 #3 %
6401      \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
6402        \shapemode\tw@
6403      \fi}}
6404  {}
6405 \IfBabelLayout{graphics}
6406  {\let\bbl@pictresetdir\relax
6407   \def\bbl@pictsetdir#1{%
6408     \ifcase\bbl@thetextdir
6409       \let\bbl@pictresetdir\relax
6410     \else
6411       \ifcase#1\bodydir TLT  % Remember this sets the inner boxes
6412         \or\textdir TLT
6413         \else\bodydir TLT \textdir TLT
6414       \fi
6415       % \(text|par)dir required in pgf:
6416       \def\bbl@pictresetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}%
6417     \fi}%
6418   \AddToHook{env/picture/begin}{\bbl@pictsetdir\tw@}%
6419   \directlua{
6420     Babel.get_picture_dir = true
6421     Babel.picture_has_bidi = 0
6422     %
6423     function Babel.picture_dir (head)
6424       if not Babel.get_picture_dir then return head end
6425       if Babel.hlist_has_bidi(head) then
6426         Babel.picture_has_bidi = 1
6427       end
6428       return head
6429     end
6430     luatexbase.add_to_callback("hpack_filter", Babel.picture_dir,
6431       "Babel.picture_dir")
6432   }%
6433   \AtBeginDocument{%
6434     \def\LS@rot{%
6435       \setbox\@outputbox\vbox{%
6436         \hbox dir TLT{\rotatebox{90}{\box\@outputbox}}}}%
6437     \long\def\put(#1,#2)#3{%
6438       \@killglue
6439       % Try:
6440       \ifx\bbl@pictresetdir\relax
6441         \def\bbl@tempc{0}%
6442       \else
6443         \directlua{
6444           Babel.get_picture_dir = true
6445           Babel.picture_has_bidi = 0
6446         }%
6447         \setbox\z@\hb@xt@\z@{%
6448           \@defaultunitsset\@tempdimc{#1}\unitlength
6449           \kern\@tempdimc
6450           #3\hss}% TODO: #3 executed twice (below). That's bad.
6451         \edef\bbl@tempc{\directlua{tex.print(Babel.picture_has_bidi)}}}%
6452       \fi
6453       % Do:
6454       \@defaultunitsset\@tempdimc{#2}\unitlength
6455       \raise\@tempdimc\hb@xt@\z@{%
6456         \@defaultunitsset\@tempdimc{#1}\unitlength
6457         \kern\@tempdimc
6458         {\ifnum\bbl@tempc>\z@\bbl@pictresetdir\fi#3}\hss}%
6459       \ignorespaces}%
6460     \MakeRobust\put}%
```

```
6461    \AtBeginDocument
6462      {\AddToHook{cmd/diagbox@pict/before}{\let\bbl@pictsetdir\@gobble}%
6463       \ifx\pgfpicture\@undefined\else % TODO. Allow deactivate?
6464         \AddToHook{env/pgfpicture/begin}{\bbl@pictsetdir\@ne}%
6465         \bbl@add\pgfinterruptpicture{\bbl@pictresetdir}%
6466         \bbl@add\pgfsys@beginpicture{\bbl@pictsetdir\z@}%
6467       \fi
6468       \ifx\tikzpicture\@undefined\else
6469         \AddToHook{env/tikzpicture/begin}{\bbl@pictsetdir\z@}%
6470         \bbl@add\tikz@atbegin@node{\bbl@pictresetdir}%
6471         \bbl@sreplace\tikz{\begingroup}{\begingroup\bbl@pictsetdir\tw@}%
6472       \fi
6473       \ifx\tcolorbox\@undefined\else
6474         \def\tcb@drawing@env@begin{%
6475         \csname tcb@before@\tcb@split@state\endcsname
6476         \bbl@pictsetdir\tw@
6477         \begin{kvtcb@graphenv}%
6478         \tcb@bbdraw%
6479         \tcb@apply@graph@patches
6480         }%
6481       \def\tcb@drawing@env@end{%
6482       \end{kvtcb@graphenv}%
6483       \bbl@pictresetdir
6484       \csname tcb@after@\tcb@split@state\endcsname
6485       }%
6486       \fi
6487     }}
6488   {}
```

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`.

```
6489  \IfBabelLayout{counters*}%
6490    {\bbl@add\bbl@opt@layout{.counters.}%
6491     \directlua{
6492       luatexbase.add_to_callback("process_output_buffer",
6493         Babel.discard_sublr , "Babel.discard_sublr") }%
6494  }{}
6495  \IfBabelLayout{counters}%
6496    {\let\bbl@OL@@textsuperscript\@textsuperscript
6497     \bbl@sreplace\@textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
6498     \let\bbl@latinarabic=\@arabic
6499     \let\bbl@OL@@arabic\@arabic
6500     \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
6501     \@ifpackagewith{babel}{bidi=default}%
6502       {\let\bbl@asciiroman=\@roman
6503        \let\bbl@OL@@roman\@roman
6504        \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciiroman#1}}}%
6505        \let\bbl@asciiRoman=\@Roman
6506        \let\bbl@OL@@roman\@Roman
6507        \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}%
6508        \let\bbl@OL@labelenumii\labelenumii
6509        \def\labelenumii{)\theenumii(}%
6510        \let\bbl@OL@p@enumiii\p@enumiii
6511        \def\p@enumiii{\p@enumii)\theenumii(}}{}}{}
6512  ⟨⟨Footnote changes⟩⟩
6513  \IfBabelLayout{footnotes}%
6514    {\let\bbl@OL@footnote\footnote
6515     \BabelFootnote\footnote\languagename{}{}%
6516     \BabelFootnote\localfootnote\languagename{}{}%
6517     \BabelFootnote\mainfootnote{}{}{}}
6518  {}
```

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.

```
6519 \IfBabelLayout{extras}%
6520   {\let\bbl@OL@underline\underline
6521    \bbl@sreplace\underline{$\@@underline}{\bbl@nextfake$\@@underline}%
6522    \let\bbl@OL@LaTeX2e\LaTeX2e
6523    \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
6524      \if b\expandafter\@car\f@series\@nil\boldmath\fi
6525      \babelsublr{%
6526        \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}}}
6527   {}
6528 ⟨/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.

```
6529 ⟨*transforms⟩
6530 Babel.linebreaking.replacements = {}
6531 Babel.linebreaking.replacements[0] = {}  -- pre
6532 Babel.linebreaking.replacements[1] = {}  -- post
6533 Babel.linebreaking.replacements[2] = {}  -- post-line WIP
6534
6535 -- Discretionaries contain strings as nodes
6536 function Babel.str_to_nodes(fn, matches, base)
6537   local n, head, last
6538   if fn == nil then return nil end
6539   for s in string.utfvalues(fn(matches)) do
6540     if base.id == 7 then
6541       base = base.replace
6542     end
6543     n = node.copy(base)
6544     n.char    = s
6545     if not head then
6546       head = n
6547     else
6548       last.next = n
6549     end
6550     last = n
6551   end
6552   return head
6553 end
6554
6555 Babel.fetch_subtext = {}
6556
6557 Babel.ignore_pre_char = function(node)
6558   return (node.lang == Babel.nohyphenation)
6559 end
6560
6561 -- Merging both functions doesn't seen feasible, because there are too
6562 -- many differences.
6563 Babel.fetch_subtext[0] = function(head)
6564   local word_string = ''
6565   local word_nodes = {}
```

```
6566    local lang
6567    local item = head
6568    local inmath = false
6569
6570    while item do
6571
6572      if item.id == 11 then
6573        inmath = (item.subtype == 0)
6574      end
6575
6576      if inmath then
6577        -- pass
6578
6579      elseif item.id == 29 then
6580        local locale = node.get_attribute(item, Babel.attr_locale)
6581
6582        if lang == locale or lang == nil then
6583          lang = lang or locale
6584          if Babel.ignore_pre_char(item) then
6585            word_string = word_string .. Babel.us_char
6586          else
6587            word_string = word_string .. unicode.utf8.char(item.char)
6588          end
6589          word_nodes[#word_nodes+1] = item
6590        else
6591          break
6592        end
6593
6594      elseif item.id == 12 and item.subtype == 13 then
6595        word_string = word_string .. ' '
6596        word_nodes[#word_nodes+1] = item
6597
6598      -- Ignore leading unrecognized nodes, too.
6599      elseif word_string ~= '' then
6600        word_string = word_string .. Babel.us_char
6601        word_nodes[#word_nodes+1] = item  -- Will be ignored
6602      end
6603
6604      item = item.next
6605    end
6606
6607    -- Here and above we remove some trailing chars but not the
6608    -- corresponding nodes. But they aren't accessed.
6609    if word_string:sub(-1) == ' ' then
6610      word_string = word_string:sub(1,-2)
6611    end
6612    word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
6613    return word_string, word_nodes, item, lang
6614 end
6615
6616 Babel.fetch_subtext[1] = function(head)
6617    local word_string = ''
6618    local word_nodes = {}
6619    local lang
6620    local item = head
6621    local inmath = false
6622
6623    while item do
6624
6625      if item.id == 11 then
6626        inmath = (item.subtype == 0)
6627      end
6628
```

```
6629      if inmath then
6630        -- pass
6631
6632      elseif item.id == 29 then
6633        if item.lang == lang or lang == nil then
6634          if (item.char ~= 124) and (item.char ~= 61) then -- not =, not |
6635            lang = lang or item.lang
6636            word_string = word_string .. unicode.utf8.char(item.char)
6637            word_nodes[#word_nodes+1] = item
6638          end
6639        else
6640          break
6641        end
6642
6643      elseif item.id == 7 and item.subtype == 2 then
6644        word_string = word_string .. '='
6645        word_nodes[#word_nodes+1] = item
6646
6647      elseif item.id == 7 and item.subtype == 3 then
6648        word_string = word_string .. '|'
6649        word_nodes[#word_nodes+1] = item
6650
6651      -- (1) Go to next word if nothing was found, and (2) implicitly
6652      -- remove leading USs.
6653      elseif word_string == '' then
6654        -- pass
6655
6656      -- This is the responsible for splitting by words.
6657      elseif (item.id == 12 and item.subtype == 13) then
6658        break
6659
6660      else
6661        word_string = word_string .. Babel.us_char
6662        word_nodes[#word_nodes+1] = item  -- Will be ignored
6663      end
6664
6665      item = item.next
6666    end
6667
6668    word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
6669    return word_string, word_nodes, item, lang
6670 end
6671
6672 function Babel.pre_hyphenate_replace(head)
6673    Babel.hyphenate_replace(head, 0)
6674 end
6675
6676 function Babel.post_hyphenate_replace(head)
6677    Babel.hyphenate_replace(head, 1)
6678 end
6679
6680 Babel.us_char = string.char(31)
6681
6682 function Babel.hyphenate_replace(head, mode)
6683    local u = unicode.utf8
6684    local lbkr = Babel.linebreaking.replacements[mode]
6685    if mode == 2 then mode = 0 end -- WIP
6686
6687    local word_head = head
6688
6689    while true do  -- for each subtext block
6690
6691      local w, w_nodes, nw, lang = Babel.fetch_subtext[mode](word_head)
```

```
6692
6693    if Babel.debug then
6694      print()
6695      print((mode == 0) and '@@@@<' or '@@@@>', w)
6696    end
6697
6698    if nw == nil and w == '' then break end
6699
6700    if not lang then goto next end
6701    if not lbkr[lang] then goto next end
6702
6703    -- For each saved (pre|post)hyphenation. TODO. Reconsider how
6704    -- loops are nested.
6705    for k=1, #lbkr[lang] do
6706      local p = lbkr[lang][k].pattern
6707      local r = lbkr[lang][k].replace
6708      local attr = lbkr[lang][k].attr or -1
6709
6710      if Babel.debug then
6711        print('*****', p, mode)
6712      end
6713
6714      -- This variable is set in some cases below to the first *byte*
6715      -- after the match, either as found by u.match (faster) or the
6716      -- computed position based on sc if w has changed.
6717      local last_match = 0
6718      local step = 0
6719
6720      -- For every match.
6721      while true do
6722        if Babel.debug then
6723          print('=====')
6724        end
6725        local new  -- used when inserting and removing nodes
6726
6727        local matches = { u.match(w, p, last_match) }
6728
6729        if #matches < 2 then break end
6730
6731        -- Get and remove empty captures (with ()'s, which return a
6732        -- number with the position), and keep actual captures
6733        -- (from (...)), if any, in matches.
6734        local first = table.remove(matches, 1)
6735        local last  = table.remove(matches, #matches)
6736        -- Non re-fetched substrings may contain \31, which separates
6737        -- subsubstrings.
6738        if string.find(w:sub(first, last-1), Babel.us_char) then break end
6739
6740        local save_last = last -- with A()BC()D, points to D
6741
6742        -- Fix offsets, from bytes to unicode. Explained above.
6743        first = u.len(w:sub(1, first-1)) + 1
6744        last  = u.len(w:sub(1, last-1)) -- now last points to C
6745
6746        -- This loop stores in a small table the nodes
6747        -- corresponding to the pattern. Used by 'data' to provide a
6748        -- predictable behavior with 'insert' (w_nodes is modified on
6749        -- the fly), and also access to 'remove'd nodes.
6750        local sc = first-1            -- Used below, too
6751        local data_nodes = {}
6752
6753        local enabled = true
6754        for q = 1, last-first+1 do
```

```
6755            data_nodes[q] = w_nodes[sc+q]
6756            if enabled
6757              and attr > -1
6758              and not node.has_attribute(data_nodes[q], attr)
6759            then
6760              enabled = false
6761            end
6762          end
6763
6764          -- This loop traverses the matched substring and takes the
6765          -- corresponding action stored in the replacement list.
6766          -- sc = the position in substr nodes / string
6767          -- rc = the replacement table index
6768          local rc = 0
6769
6770          while rc < last-first+1 do -- for each replacement
6771            if Babel.debug then
6772              print('.....', rc + 1)
6773            end
6774            sc = sc + 1
6775            rc = rc + 1
6776
6777            if Babel.debug then
6778              Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6779              local ss = ''
6780              for itt in node.traverse(head) do
6781                if itt.id == 29 then
6782                  ss = ss .. unicode.utf8.char(itt.char)
6783                else
6784                  ss = ss .. '{' .. itt.id .. '}'
6785                end
6786              end
6787              print('*****************', ss)
6788
6789            end
6790
6791            local crep = r[rc]
6792            local item = w_nodes[sc]
6793            local item_base = item
6794            local placeholder = Babel.us_char
6795            local d
6796
6797            if crep and crep.data then
6798              item_base = data_nodes[crep.data]
6799            end
6800
6801            if crep then
6802              step = crep.step or 0
6803            end
6804
6805            if (not enabled) or (crep and next(crep) == nil) then -- = {}
6806              last_match = save_last    -- Optimization
6807              goto next
6808
6809            elseif crep == nil or crep.remove then
6810              node.remove(head, item)
6811              table.remove(w_nodes, sc)
6812              w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
6813              sc = sc - 1  -- Nothing has been inserted.
6814              last_match = utf8.offset(w, sc+1+step)
6815              goto next
6816
6817            elseif crep and crep.kashida then -- Experimental
```

195

```
6818              node.set_attribute(item,
6819                Babel.attr_kashida,
6820                crep.kashida)
6821            last_match = utf8.offset(w, sc+1+step)
6822            goto next
6823
6824        elseif crep and crep.string then
6825          local str = crep.string(matches)
6826          if str == '' then  -- Gather with nil
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          else
6832            local loop_first = true
6833            for s in string.utfvalues(str) do
6834              d = node.copy(item_base)
6835              d.char = s
6836              if loop_first then
6837                loop_first = false
6838                head, new = node.insert_before(head, item, d)
6839                if sc == 1 then
6840                  word_head = head
6841                end
6842                w_nodes[sc] = d
6843                w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc+1)
6844              else
6845                sc = sc + 1
6846                head, new = node.insert_before(head, item, d)
6847                table.insert(w_nodes, sc, new)
6848                w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc)
6849              end
6850              if Babel.debug then
6851                print('.....', 'str')
6852                Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6853              end
6854            end  -- for
6855            node.remove(head, item)
6856          end  -- if ''
6857          last_match = utf8.offset(w, sc+1+step)
6858          goto next
6859
6860        elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
6861          d = node.new(7, 0)   -- (disc, discretionary)
6862          d.pre     = Babel.str_to_nodes(crep.pre, matches, item_base)
6863          d.post    = Babel.str_to_nodes(crep.post, matches, item_base)
6864          d.replace = Babel.str_to_nodes(crep.no, matches, item_base)
6865          d.attr = item_base.attr
6866          if crep.pre == nil then  -- TeXbook p96
6867            d.penalty = crep.penalty or tex.hyphenpenalty
6868          else
6869            d.penalty = crep.penalty or tex.exhyphenpenalty
6870          end
6871          placeholder = '|'
6872          head, new = node.insert_before(head, item, d)
6873
6874        elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
6875          -- ERROR
6876
6877        elseif crep and crep.penalty then
6878          d = node.new(14, 0)   -- (penalty, userpenalty)
6879          d.attr = item_base.attr
6880          d.penalty = crep.penalty
```

```
6881            head, new = node.insert_before(head, item, d)
6882
6883         elseif crep and crep.space then
6884            -- 655360 = 10 pt = 10 * 65536 sp
6885            d = node.new(12, 13)       -- (glue, spaceskip)
6886            local quad = font.getfont(item_base.font).size or 655360
6887            node.setglue(d, crep.space[1] * quad,
6888                            crep.space[2] * quad,
6889                            crep.space[3] * quad)
6890            if mode == 0 then
6891               placeholder = ' '
6892            end
6893            head, new = node.insert_before(head, item, d)
6894
6895         elseif crep and crep.spacefactor then
6896            d = node.new(12, 13)       -- (glue, spaceskip)
6897            local base_font = font.getfont(item_base.font)
6898            node.setglue(d,
6899               crep.spacefactor[1] * base_font.parameters['space'],
6900               crep.spacefactor[2] * base_font.parameters['space_stretch'],
6901               crep.spacefactor[3] * base_font.parameters['space_shrink'])
6902            if mode == 0 then
6903               placeholder = ' '
6904            end
6905            head, new = node.insert_before(head, item, d)
6906
6907         elseif mode == 0 and crep and crep.space then
6908            -- ERROR
6909
6910         end  -- ie replacement cases
6911
6912         -- Shared by disc, space and penalty.
6913         if sc == 1 then
6914            word_head = head
6915         end
6916         if crep.insert then
6917            w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc)
6918            table.insert(w_nodes, sc, new)
6919            last = last + 1
6920         else
6921            w_nodes[sc] = d
6922            node.remove(head, item)
6923            w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1)
6924         end
6925
6926         last_match = utf8.offset(w, sc+1+step)
6927
6928         ::next::
6929
6930      end  -- for each replacement
6931
6932      if Babel.debug then
6933         print('.....', '/')
6934         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6935      end
6936
6937   end  -- for match
6938
6939 end  -- for patterns
6940
6941 ::next::
6942 word_head = nw
6943 end  -- for substring
```

```
6944   return head
6945 end
6946
6947 -- This table stores capture maps, numbered consecutively
6948 Babel.capture_maps = {}
6949
6950 -- The following functions belong to the next macro
6951 function Babel.capture_func(key, cap)
6952   local ret = "[[" .. cap:gsub('{([0-9])}', "]]..m[%1]..[[") .. "]]"
6953   local cnt
6954   local u = unicode.utf8
6955   ret, cnt = ret:gsub('{([0-9])|([^|]+)|(.-)}', Babel.capture_func_map)
6956   if cnt == 0 then
6957     ret = u.gsub(ret, '{(%x%x%x%x+)}',
6958           function (n)
6959             return u.char(tonumber(n, 16))
6960           end)
6961   end
6962   ret = ret:gsub("%[%[%]%]%.%.", '')
6963   ret = ret:gsub("%.%.%[%[%]%]", '')
6964   return key .. [[=function(m) return ]] .. ret .. [[ end]]
6965 end
6966
6967 function Babel.capt_map(from, mapno)
6968   return Babel.capture_maps[mapno][from] or from
6969 end
6970
6971 -- Handle the {n|abc|ABC} syntax in captures
6972 function Babel.capture_func_map(capno, from, to)
6973   local u = unicode.utf8
6974   from = u.gsub(from, '{(%x%x%x%x+)}',
6975         function (n)
6976           return u.char(tonumber(n, 16))
6977         end)
6978   to = u.gsub(to, '{(%x%x%x%x+)}',
6979         function (n)
6980           return u.char(tonumber(n, 16))
6981         end)
6982   local froms = {}
6983   for s in string.utfcharacters(from) do
6984     table.insert(froms, s)
6985   end
6986   local cnt = 1
6987   table.insert(Babel.capture_maps, {})
6988   local mlen = table.getn(Babel.capture_maps)
6989   for s in string.utfcharacters(to) do
6990     Babel.capture_maps[mlen][froms[cnt]] = s
6991     cnt = cnt + 1
6992   end
6993   return "]]..Babel.capt_map(m[" .. capno .. "]," ..
6994         (mlen) .. ").." .. "[["
6995 end
6996
6997 -- Create/Extend reversed sorted list of kashida weights:
6998 function Babel.capture_kashida(key, wt)
6999   wt = tonumber(wt)
7000   if Babel.kashida_wts then
7001     for p, q in ipairs(Babel.kashida_wts) do
7002       if wt  == q then
7003         break
7004       elseif wt > q then
7005         table.insert(Babel.kashida_wts, p, wt)
7006         break
```

```
7007      elseif table.getn(Babel.kashida_wts) == p then
7008        table.insert(Babel.kashida_wts, wt)
7009      end
7010    end
7011  else
7012    Babel.kashida_wts = { wt }
7013  end
7014  return 'kashida = ' .. wt
7015 end
7016 ⟨/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.

```
7017 ⟨∗basic-r⟩
7018 Babel = Babel or {}
7019
7020 Babel.bidi_enabled = true
7021
7022 require('babel-data-bidi.lua')
7023
7024 local characters = Babel.characters
7025 local ranges = Babel.ranges
7026
7027 local DIR = node.id("dir")
7028
7029 local function dir_mark(head, from, to, outer)
7030   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
```

```
7031  local d = node.new(DIR)
7032  d.dir = '+' .. dir
7033  node.insert_before(head, from, d)
7034  d = node.new(DIR)
7035  d.dir = '-' .. dir
7036  node.insert_after(head, to, d)
7037 end
7038
7039 function Babel.bidi(head, ispar)
7040   local first_n, last_n        -- first and last char with nums
7041   local last_es                -- an auxiliary 'last' used with nums
7042   local first_d, last_d        -- first and last char in L/R block
7043   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):

```
7044   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
7045   local strong_lr = (strong == 'l') and 'l' or 'r'
7046   local outer = strong
7047
7048   local new_dir = false
7049   local first_dir = false
7050   local inmath = false
7051
7052   local last_lr
7053
7054   local type_n = ''
7055
7056   for item in node.traverse(head) do
7057
7058     -- three cases: glyph, dir, otherwise
7059     if item.id == node.id'glyph'
7060       or (item.id == 7 and item.subtype == 2) then
7061
7062       local itemchar
7063       if item.id == 7 and item.subtype == 2 then
7064         itemchar = item.replace.char
7065       else
7066         itemchar = item.char
7067       end
7068       local chardata = characters[itemchar]
7069       dir = chardata and chardata.d or nil
7070       if not dir then
7071         for nn, et in ipairs(ranges) do
7072           if itemchar < et[1] then
7073             break
7074           elseif itemchar <= et[2] then
7075             dir = et[3]
7076             break
7077           end
7078         end
7079       end
7080       dir = dir or 'l'
7081       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).

```
7082       if new_dir then
7083         attr_dir = 0
7084         for at in node.traverse(item.attr) do
```

```
7085          if at.number == Babel.attr_dir then
7086            attr_dir = at.value & 0x3
7087          end
7088        end
7089        if attr_dir == 1 then
7090          strong = 'r'
7091        elseif attr_dir == 2 then
7092          strong = 'al'
7093        else
7094          strong = 'l'
7095        end
7096        strong_lr = (strong == 'l') and 'l' or 'r'
7097        outer = strong_lr
7098        new_dir = false
7099      end
7100
7101      if dir == 'nsm' then dir = strong end              -- W1
```

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

```
7102      dir_real = dir            -- We need dir_real to set strong below
7103      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:

```
7104      if strong == 'al' then
7105        if dir == 'en' then dir = 'an' end              -- W2
7106        if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
7107        strong_lr = 'r'                                  -- W3
7108      end
```

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

```
7109    elseif item.id == node.id'dir' and not inmath then
7110      new_dir = true
7111      dir = nil
7112    elseif item.id == node.id'math' then
7113      inmath = (item.subtype == 0)
7114    else
7115      dir = nil            -- Not a char
7116    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>.

```
7117    if dir == 'en' or dir == 'an' or dir == 'et' then
7118      if dir ~= 'et' then
7119        type_n = dir
7120      end
7121      first_n = first_n or item
7122      last_n = last_es or item
7123      last_es = nil
7124    elseif dir == 'es' and last_n then -- W3+W6
7125      last_es = item
7126    elseif dir == 'cs' then              -- it's right - do nothing
7127    elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
7128      if strong_lr == 'r' and type_n ~= '' then
7129        dir_mark(head, first_n, last_n, 'r')
7130      elseif strong_lr == 'l' and first_d and type_n == 'an' then
7131        dir_mark(head, first_n, last_n, 'r')
7132        dir_mark(head, first_d, last_d, outer)
7133        first_d, last_d = nil, nil
7134      elseif strong_lr == 'l' and type_n ~= '' then
7135        last_d = last_n
```

201

```
7136        end
7137        type_n = ''
7138        first_n, last_n = nil, nil
7139      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:

```
7140      if dir == 'l' or dir == 'r' then
7141        if dir ~= outer then
7142          first_d = first_d or item
7143          last_d = item
7144        elseif first_d and dir ~= strong_lr then
7145          dir_mark(head, first_d, last_d, outer)
7146          first_d, last_d = nil, nil
7147        end
7148      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.

```
7149      if dir and not last_lr and dir ~= 'l' and outer == 'r' then
7150        item.char = characters[item.char] and
7151                    characters[item.char].m or item.char
7152      elseif (dir or new_dir) and last_lr ~= item then
7153        local mir = outer .. strong_lr .. (dir or outer)
7154        if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
7155          for ch in node.traverse(node.next(last_lr)) do
7156            if ch == item then break end
7157            if ch.id == node.id'glyph' and characters[ch.char] then
7158              ch.char = characters[ch.char].m or ch.char
7159            end
7160          end
7161        end
7162      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).

```
7163      if dir == 'l' or dir == 'r' then
7164        last_lr = item
7165        strong = dir_real              -- Don't search back - best save now
7166        strong_lr = (strong == 'l') and 'l' or 'r'
7167      elseif new_dir then
7168        last_lr = nil
7169      end
7170    end
```

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

```
7171    if last_lr and outer == 'r' then
7172      for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
7173        if characters[ch.char] then
7174          ch.char = characters[ch.char].m or ch.char
7175        end
7176      end
7177    end
7178    if first_n then
7179      dir_mark(head, first_n, last_n, outer)
7180    end
7181    if first_d then
7182      dir_mark(head, first_d, last_d, outer)
7183    end
```

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

```
7184    return node.prev(head) or head
7185 end
7186 ⟨/basic-r⟩
```

And here the Lua code for `bidi=basic`:

```
7187 ⟨*basic⟩
7188 Babel = Babel or {}
7189
7190 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
7191
7192 Babel.fontmap = Babel.fontmap or {}
7193 Babel.fontmap[0] = {}        -- l
7194 Babel.fontmap[1] = {}        -- r
7195 Babel.fontmap[2] = {}        -- al/an
7196
7197 Babel.bidi_enabled = true
7198 Babel.mirroring_enabled = true
7199
7200 require('babel-data-bidi.lua')
7201
7202 local characters = Babel.characters
7203 local ranges = Babel.ranges
7204
7205 local DIR = node.id('dir')
7206 local GLYPH = node.id('glyph')
7207
7208 local function insert_implicit(head, state, outer)
7209   local new_state = state
7210   if state.sim and state.eim and state.sim ~= state.eim then
7211     dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
7212     local d = node.new(DIR)
7213     d.dir = '+' .. dir
7214     node.insert_before(head, state.sim, d)
7215     local d = node.new(DIR)
7216     d.dir = '-' .. dir
7217     node.insert_after(head, state.eim, d)
7218   end
7219   new_state.sim, new_state.eim = nil, nil
7220   return head, new_state
7221 end
7222
7223 local function insert_numeric(head, state)
7224   local new
7225   local new_state = state
7226   if state.san and state.ean and state.san ~= state.ean then
7227     local d = node.new(DIR)
7228     d.dir = '+TLT'
7229     _, new = node.insert_before(head, state.san, d)
7230     if state.san == state.sim then state.sim = new end
7231     local d = node.new(DIR)
7232     d.dir = '-TLT'
7233     _, new = node.insert_after(head, state.ean, d)
7234     if state.ean == state.eim then state.eim = new end
7235   end
7236   new_state.san, new_state.ean = nil, nil
7237   return head, new_state
7238 end
7239
7240 -- TODO - \hbox with an explicit dir can lead to wrong results
7241 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
7242 -- was s made to improve the situation, but the problem is the 3-dir
```

203

```
7243 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
7244 -- well.
7245
7246 function Babel.bidi(head, ispar, hdir)
7247   local d    -- d is used mainly for computations in a loop
7248   local prev_d = ''
7249   local new_d = false
7250
7251   local nodes = {}
7252   local outer_first = nil
7253   local inmath = false
7254
7255   local glue_d = nil
7256   local glue_i = nil
7257
7258   local has_en = false
7259   local first_et = nil
7260
7261   local has_hyperlink = false
7262
7263   local ATDIR = Babel.attr_dir
7264
7265   local save_outer
7266   local temp = node.get_attribute(head, ATDIR)
7267   if temp then
7268     temp = temp & 0x3
7269     save_outer = (temp == 0 and 'l') or
7270                  (temp == 1 and 'r') or
7271                  (temp == 2 and 'al')
7272   elseif ispar then          -- Or error? Shouldn't happen
7273     save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
7274   else                       -- Or error? Shouldn't happen
7275     save_outer = ('TRT' == hdir) and 'r' or 'l'
7276   end
7277     -- when the callback is called, we are just _after_ the box,
7278     -- and the textdir is that of the surrounding text
7279   -- if not ispar and hdir ~= tex.textdir then
7280   --   save_outer = ('TRT' == hdir) and 'r' or 'l'
7281   -- end
7282   local outer = save_outer
7283   local last = outer
7284   -- 'al' is only taken into account in the first, current loop
7285   if save_outer == 'al' then save_outer = 'r' end
7286
7287   local fontmap = Babel.fontmap
7288
7289   for item in node.traverse(head) do
7290
7291     -- In what follows, #node is the last (previous) node, because the
7292     -- current one is not added until we start processing the neutrals.
7293
7294     -- three cases: glyph, dir, otherwise
7295     if item.id == GLYPH
7296       or (item.id == 7 and item.subtype == 2) then
7297
7298       local d_font = nil
7299       local item_r
7300       if item.id == 7 and item.subtype == 2 then
7301         item_r = item.replace    -- automatic discs have just 1 glyph
7302       else
7303         item_r = item
7304       end
7305       local chardata = characters[item_r.char]
```

204

```
7306        d = chardata and chardata.d or nil
7307        if not d or d == 'nsm' then
7308          for nn, et in ipairs(ranges) do
7309            if item_r.char < et[1] then
7310              break
7311            elseif item_r.char <= et[2] then
7312              if not d then d = et[3]
7313              elseif d == 'nsm' then d_font = et[3]
7314              end
7315              break
7316            end
7317          end
7318        end
7319        d = d or 'l'
7320
7321        -- A short 'pause' in bidi for mapfont
7322        d_font = d_font or d
7323        d_font = (d_font == 'l' and 0) or
7324                 (d_font == 'nsm' and 0) or
7325                 (d_font == 'r' and 1) or
7326                 (d_font == 'al' and 2) or
7327                 (d_font == 'an' and 2) or nil
7328        if d_font and fontmap and fontmap[d_font][item_r.font] then
7329          item_r.font = fontmap[d_font][item_r.font]
7330        end
7331
7332        if new_d then
7333          table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
7334          if inmath then
7335            attr_d = 0
7336          else
7337            attr_d = node.get_attribute(item, ATDIR)
7338            attr_d = attr_d & 0x3
7339          end
7340          if attr_d == 1 then
7341            outer_first = 'r'
7342            last = 'r'
7343          elseif attr_d == 2 then
7344            outer_first = 'r'
7345            last = 'al'
7346          else
7347            outer_first = 'l'
7348            last = 'l'
7349          end
7350          outer = last
7351          has_en = false
7352          first_et = nil
7353          new_d = false
7354        end
7355
7356        if glue_d then
7357          if (d == 'l' and 'l' or 'r') ~= glue_d then
7358            table.insert(nodes, {glue_i, 'on', nil})
7359          end
7360          glue_d = nil
7361          glue_i = nil
7362        end
7363
7364      elseif item.id == DIR then
7365        d = nil
7366
7367        if head ~= item then new_d = true end
7368
```

205

```
7369    elseif item.id == node.id'glue' and item.subtype == 13 then
7370      glue_d = d
7371      glue_i = item
7372      d = nil
7373
7374    elseif item.id == node.id'math' then
7375      inmath = (item.subtype == 0)
7376
7377    elseif item.id == 8 and item.subtype == 19 then
7378      has_hyperlink = true
7379
7380    else
7381      d = nil
7382    end
7383
7384    -- AL <= EN/ET/ES      -- W2 + W3 + W6
7385    if last == 'al' and d == 'en' then
7386      d = 'an'              -- W3
7387    elseif last == 'al' and (d == 'et' or d == 'es') then
7388      d = 'on'              -- W6
7389    end
7390
7391    -- EN + CS/ES + EN      -- W4
7392    if d == 'en' and #nodes >= 2 then
7393      if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
7394          and nodes[#nodes-1][2] == 'en' then
7395        nodes[#nodes][2] = 'en'
7396      end
7397    end
7398
7399    -- AN + CS + AN         -- W4 too, because uax9 mixes both cases
7400    if d == 'an' and #nodes >= 2 then
7401      if (nodes[#nodes][2] == 'cs')
7402          and nodes[#nodes-1][2] == 'an' then
7403        nodes[#nodes][2] = 'an'
7404      end
7405    end
7406
7407    -- ET/EN                -- W5 + W7->l / W6->on
7408    if d == 'et' then
7409      first_et = first_et or (#nodes + 1)
7410    elseif d == 'en' then
7411      has_en = true
7412      first_et = first_et or (#nodes + 1)
7413    elseif first_et then          -- d may be nil here !
7414      if has_en then
7415        if last == 'l' then
7416          temp = 'l'     -- W7
7417        else
7418          temp = 'en'    -- W5
7419        end
7420      else
7421        temp = 'on'      -- W6
7422      end
7423      for e = first_et, #nodes do
7424        if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
7425      end
7426      first_et = nil
7427      has_en = false
7428    end
7429
7430    -- Force mathdir in math if ON (currently works as expected only
7431    -- with 'l')
```

```
7432    if inmath and d == 'on' then
7433      d = ('TRT' == tex.mathdir) and 'r' or 'l'
7434    end
7435
7436    if d then
7437      if d == 'al' then
7438        d = 'r'
7439        last = 'al'
7440      elseif d == 'l' or d == 'r' then
7441        last = d
7442      end
7443      prev_d = d
7444      table.insert(nodes, {item, d, outer_first})
7445    end
7446
7447    outer_first = nil
7448
7449  end
7450
7451  -- TODO -- repeated here in case EN/ET is the last node. Find a
7452  -- better way of doing things:
7453  if first_et then        -- dir may be nil here !
7454    if has_en then
7455      if last == 'l' then
7456        temp = 'l'     -- W7
7457      else
7458        temp = 'en'    -- W5
7459      end
7460    else
7461      temp = 'on'      -- W6
7462    end
7463    for e = first_et, #nodes do
7464      if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
7465    end
7466  end
7467
7468  -- dummy node, to close things
7469  table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
7470
7471  --------------  NEUTRAL -----------------
7472
7473  outer = save_outer
7474  last = outer
7475
7476  local first_on = nil
7477
7478  for q = 1, #nodes do
7479    local item
7480
7481    local outer_first = nodes[q][3]
7482    outer = outer_first or outer
7483    last = outer_first or last
7484
7485    local d = nodes[q][2]
7486    if d == 'an' or d == 'en' then d = 'r' end
7487    if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
7488
7489    if d == 'on' then
7490      first_on = first_on or q
7491    elseif first_on then
7492      if last == d then
7493        temp = d
7494      else
```

```
7495              temp = outer
7496            end
7497            for r = first_on, q - 1 do
7498              nodes[r][2] = temp
7499              item = nodes[r][1]     -- MIRRORING
7500              if Babel.mirroring_enabled and item.id == GLYPH
7501                  and temp == 'r' and characters[item.char] then
7502                local font_mode = ''
7503                if item.font > 0 and font.fonts[item.font].properties then
7504                  font_mode = font.fonts[item.font].properties.mode
7505                end
7506                if font_mode ~= 'harf' and font_mode ~= 'plug' then
7507                  item.char = characters[item.char].m or item.char
7508                end
7509              end
7510            end
7511            first_on = nil
7512          end
7513
7514          if d == 'r' or d == 'l' then last = d end
7515        end
7516
7517        -------------  IMPLICIT, REORDER ---------------
7518
7519        outer = save_outer
7520        last = outer
7521
7522        local state = {}
7523        state.has_r = false
7524
7525        for q = 1, #nodes do
7526
7527          local item = nodes[q][1]
7528
7529          outer = nodes[q][3] or outer
7530
7531          local d = nodes[q][2]
7532
7533          if d == 'nsm' then d = last end              -- W1
7534          if d == 'en' then d = 'an' end
7535          local isdir = (d == 'r' or d == 'l')
7536
7537          if outer == 'l' and d == 'an' then
7538            state.san = state.san or item
7539            state.ean = item
7540          elseif state.san then
7541            head, state = insert_numeric(head, state)
7542          end
7543
7544          if outer == 'l' then
7545            if d == 'an' or d == 'r' then     -- im -> implicit
7546              if d == 'r' then state.has_r = true end
7547              state.sim = state.sim or item
7548              state.eim = item
7549            elseif d == 'l' and state.sim and state.has_r then
7550              head, state = insert_implicit(head, state, outer)
7551            elseif d == 'l' then
7552              state.sim, state.eim, state.has_r = nil, nil, false
7553            end
7554          else
7555            if d == 'an' or d == 'l' then
7556              if nodes[q][3] then -- nil except after an explicit dir
7557                state.sim = item  -- so we move sim 'inside' the group
```

```
7558            else
7559               state.sim = state.sim or item
7560            end
7561            state.eim = item
7562          elseif d == 'r' and state.sim then
7563            head, state = insert_implicit(head, state, outer)
7564          elseif d == 'r' then
7565            state.sim, state.eim = nil, nil
7566          end
7567       end
7568
7569       if isdir then
7570         last = d              -- Don't search back - best save now
7571       elseif d == 'on' and state.san   then
7572         state.san = state.san or item
7573         state.ean = item
7574       end
7575
7576    end
7577
7578    head = node.prev(head) or head
7579
7580    -------------- FIX HYPERLINKS ----------------
7581
7582    if has_hyperlink then
7583      local flag, linking = 0, 0
7584      for item in node.traverse(head) do
7585        if item.id == DIR then
7586          if item.dir == '+TRT' or item.dir == '+TLT' then
7587            flag = flag + 1
7588          elseif item.dir == '-TRT' or item.dir == '-TLT' then
7589            flag = flag - 1
7590          end
7591        elseif item.id == 8 and item.subtype == 19 then
7592          linking = flag
7593        elseif item.id == 8 and item.subtype == 20 then
7594          if linking > 0 then
7595            if item.prev.id == DIR and
7596                (item.prev.dir == '-TRT' or item.prev.dir == '-TLT') then
7597              d = node.new(DIR)
7598              d.dir = item.prev.dir
7599              node.remove(head, item.prev)
7600              node.insert_after(head, item, d)
7601            end
7602          end
7603          linking = 0
7604        end
7605      end
7606    end
7607
7608    return head
7609 end
7610 ⟨/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.

```
7611 ⟨∗nil⟩
7612 \ProvidesLanguage{nil}[⟨⟨date⟩⟩ ⟨⟨version⟩⟩ Nil language]
7613 \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.

```
7614 \ifx\l@nil\@undefined
7615   \newlanguage\l@nil
7616   \@namedef{bbl@hyphendata@\the\l@nil}{{}{}}% Remove warning
7617   \let\bbl@elt\relax
7618   \edef\bbl@languages{%  Add it to the list of languages
7619     \bbl@languages\bbl@elt{nil}{\the\l@nil}{}{}}
7620 \fi
```

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

```
7621 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}
```

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

\captionnil
\datenil
```
7622 \let\captionsnil\@empty
7623 \let\datenil\@empty
```

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

```
7624 \def\bbl@inidata@nil{%
7625   \bbl@elt{identification}{tag.ini}{und}%
7626   \bbl@elt{identification}{load.level}{0}%
7627   \bbl@elt{identification}{charset}{utf8}%
7628   \bbl@elt{identification}{version}{1.0}%
7629   \bbl@elt{identification}{date}{2022-05-16}%
7630   \bbl@elt{identification}{name.local}{nil}%
7631   \bbl@elt{identification}{name.english}{nil}%
7632   \bbl@elt{identification}{name.babel}{nil}%
7633   \bbl@elt{identification}{tag.bcp47}{und}%
7634   \bbl@elt{identification}{language.tag.bcp47}{und}%
7635   \bbl@elt{identification}{tag.opentype}{dflt}%
7636   \bbl@elt{identification}{script.name}{Latin}%
7637   \bbl@elt{identification}{script.tag.bcp47}{Latn}%
7638   \bbl@elt{identification}{script.tag.opentype}{DFLT}%
7639   \bbl@elt{identification}{level}{1}%
7640   \bbl@elt{identification}{encodings}{}%
7641   \bbl@elt{identification}{derivate}{no}}
7642 \@namedef{bbl@tbcp@nil}{und}
7643 \@namedef{bbl@lbcp@nil}{und}
7644 \@namedef{bbl@lotf@nil}{dflt}
7645 \@namedef{bbl@elname@nil}{nil}
7646 \@namedef{bbl@lname@nil}{nil}
7647 \@namedef{bbl@esname@nil}{Latin}
7648 \@namedef{bbl@sname@nil}{Latin}
```

```
7649 \@namedef{bbl@sbcp@nil}{Latn}
7650 \@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.

```
7651 \ldf@finish{nil}
7652 ⟨/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.

```
7653 ⟨⟨*Compute Julian day⟩⟩ ≡
7654 \def\bbl@fpmod#1#2{(#1-#2*floor(#1/#2))}
7655 \def\bbl@cs@gregleap#1{%
7656   (\bbl@fpmod{#1}{4} == 0) &&
7657     (!((\bbl@fpmod{#1}{100} == 0) && (\bbl@fpmod{#1}{400} != 0)))}
7658 \def\bbl@cs@jd#1#2#3{% year, month, day
7659   \fp_eval:n{ 1721424.5   + (365 * (#1 - 1)) +
7660     floor((#1 - 1) / 4)   + (-floor((#1 - 1) / 100)) +
7661     floor((#1 - 1) / 400) + floor((((367 * #2) - 362) / 12) +
7662     ((#2 <= 2) ? 0 : (\bbl@cs@gregleap{#1} ? -1 : -2)) + #3) }}
7663 ⟨⟨/Compute Julian day⟩⟩
```

15.1 Islamic

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

```
7664 ⟨*ca-islamic⟩
7665 \ExplSyntaxOn
7666 ⟨⟨Compute Julian day⟩⟩
7667 % == islamic (default)
7668 % Not yet implemented
7669 \def\bbl@ca@islamic#1-#2-#3\@@#4#5#6{}
```

The Civil calendar.

```
7670 \def\bbl@cs@isltojd#1#2#3{ % year, month, day
7671   ((#3 + ceil(29.5 * (#2 - 1)) +
7672   (#1 - 1) * 354 + floor((3 + (11 * #1)) / 30) +
7673   1948439.5) - 1) }
7674 \@namedef{bbl@ca@islamic-civil++}{\bbl@ca@islamicvl@x{+2}}
7675 \@namedef{bbl@ca@islamic-civil+}{\bbl@ca@islamicvl@x{+1}}
7676 \@namedef{bbl@ca@islamic-civil}{\bbl@ca@islamicvl@x{}}
7677 \@namedef{bbl@ca@islamic-civil-}{\bbl@ca@islamicvl@x{-1}}
7678 \@namedef{bbl@ca@islamic-civil--}{\bbl@ca@islamicvl@x{-2}}
7679 \def\bbl@ca@islamicvl@x#1#2-#3-#4\@@#5#6#7{%
7680   \edef\bbl@tempa{%
7681     \fp_eval:n{ floor(\bbl@cs@jd{#2}{#3}{#4})+0.5 #1}}%
7682   \edef#5{%
7683     \fp_eval:n{ floor(((30*(\bbl@tempa-1948439.5)) + 10646)/10631) }}%
7684   \edef#6{\fp_eval:n{
7685     min(12,ceil((\bbl@tempa-(29+\bbl@cs@isltojd{#5}{1}{1}))/29.5)+1) }}%
7686   \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).

```
7687 \def\bbl@cs@umalqura@data{56660, 56690,56719,56749,56778,56808,%
7688   56837,56867,56897,56926,56956,56985,57015,57044,57074,57103,%
7689   57133,57162,57192,57221,57251,57280,57310,57340,57369,57399,%
```

```
7690    57429,57458,57487,57517,57546,57576,57605,57634,57664,57694,%
7691    57723,57753,57783,57813,57842,57871,57901,57930,57959,57989,%
7692    58018,58048,58077,58107,58137,58167,58196,58226,58255,58285,%
7693    58314,58343,58373,58402,58432,58461,58491,58521,58551,58580,%
7694    58610,58639,58669,58698,58727,58757,58786,58816,58845,58875,%
7695    58905,58934,58964,58994,59023,59053,59082,59111,59141,59170,%
7696    59200,59229,59259,59288,59318,59348,59377,59407,59436,59466,%
7697    59495,59525,59554,59584,59613,59643,59672,59702,59731,59761,%
7698    59791,59820,59850,59879,59909,59939,59968,59997,60027,60056,%
7699    60086,60115,60145,60174,60204,60234,60264,60293,60323,60352,%
7700    60381,60411,60440,60469,60499,60528,60558,60588,60618,60648,%
7701    60677,60707,60736,60765,60795,60824,60853,60883,60912,60942,%
7702    60972,61002,61031,61061,61090,61120,61149,61179,61208,61237,%
7703    61267,61296,61326,61356,61385,61415,61445,61474,61504,61533,%
7704    61563,61592,61621,61651,61680,61710,61739,61769,61799,61828,%
7705    61858,61888,61917,61947,61976,62006,62035,62064,62094,62123,%
7706    62153,62182,62212,62242,62271,62301,62331,62360,62390,62419,%
7707    62448,62478,62507,62537,62566,62596,62625,62655,62685,62715,%
7708    62744,62774,62803,62832,62862,62891,62921,62950,62980,63009,%
7709    63039,63069,63099,63128,63157,63187,63216,63246,63275,63305,%
7710    63334,63363,63393,63423,63453,63482,63512,63541,63571,63600,%
7711    63630,63659,63689,63718,63747,63777,63807,63836,63866,63895,%
7712    63925,63955,63984,64014,64043,64073,64102,64131,64161,64190,%
7713    64220,64249,64279,64309,64339,64368,64398,64427,64457,64486,%
7714    64515,64545,64574,64603,64633,64663,64692,64722,64752,64782,%
7715    64811,64841,64870,64899,64929,64958,64987,65017,65047,65076,%
7716    65106,65136,65166,65195,65225,65254,65283,65313,65342,65371,%
7717    65401,65431,65460,65490,65520}
7718 \@namedef{bbl@ca@islamic-umalqura+}{\bbl@ca@islamcuqr@x{+1}}
7719 \@namedef{bbl@ca@islamic-umalqura}{\bbl@ca@islamcuqr@x{}}
7720 \@namedef{bbl@ca@islamic-umalqura-}{\bbl@ca@islamcuqr@x{-1}}
7721 \def\bbl@ca@islamcuqr@x#1#2-#3-#4\@@#5#6#7{%
7722    \ifnum#2>2014 \ifnum#2<2038
7723      \bbl@afterfi\expandafter\@gobble
7724    \fi\fi
7725      {\bbl@error{Year~out~of~range}{The~allowed~range~is~2014-2038}}%
7726    \edef\bbl@tempd{\fp_eval:n{ % (Julian) day
7727      \bbl@cs@jd{#2}{#3}{#4} + 0.5 - 2400000 #1}}%
7728    \count@\@ne
7729    \bbl@foreach\bbl@cs@umalqura@data{%
7730      \advance\count@\@ne
7731      \ifnum##1>\bbl@tempd\else
7732        \edef\bbl@tempe{\the\count@}%
7733        \edef\bbl@tempb{##1}%
7734      \fi}%
7735    \edef\bbl@templ{\fp_eval:n{ \bbl@tempe + 16260 + 949 }}% month~lunar
7736    \edef\bbl@tempa{\fp_eval:n{ floor((\bbl@templ - 1 ) / 12) }}% annus
7737    \edef#5{\fp_eval:n{ \bbl@tempa + 1  }}%
7738    \edef#6{\fp_eval:n{ \bbl@templ - (12 * \bbl@tempa) }}%
7739    \edef#7{\fp_eval:n{ \bbl@tempd - \bbl@tempb + 1 }}}
7740 \ExplSyntaxOff
7741 \bbl@add\bbl@precalendar{%
7742    \bbl@replace\bbl@ld@calendar{-civil}{}%
7743    \bbl@replace\bbl@ld@calendar{-umalqura}{}%
7744    \bbl@replace\bbl@ld@calendar{+}{}%
7745    \bbl@replace\bbl@ld@calendar{-}{}}
7746 ⟨/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`

```
7747 ⟨∗ca-hebrew⟩
7748 \newcount\bbl@cntcommon
7749 \def\bbl@remainder#1#2#3{%
7750   #3=#1\relax
7751   \divide #3 by #2\relax
7752   \multiply #3 by -#2\relax
7753   \advance #3 by #1\relax}%
7754 \newif\ifbbl@divisible
7755 \def\bbl@checkifdivisible#1#2{%
7756   {\countdef\tmp=0
7757   \bbl@remainder{#1}{#2}{\tmp}%
7758   \ifnum \tmp=0
7759       \global\bbl@divisibletrue
7760   \else
7761       \global\bbl@divisiblefalse
7762   \fi}}
7763 \newif\ifbbl@gregleap
7764 \def\bbl@ifgregleap#1{%
7765   \bbl@checkifdivisible{#1}{4}%
7766   \ifbbl@divisible
7767       \bbl@checkifdivisible{#1}{100}%
7768       \ifbbl@divisible
7769           \bbl@checkifdivisible{#1}{400}%
7770           \ifbbl@divisible
7771               \bbl@gregleaptrue
7772           \else
7773               \bbl@gregleapfalse
7774           \fi
7775       \else
7776           \bbl@gregleaptrue
7777       \fi
7778   \else
7779       \bbl@gregleapfalse
7780   \fi
7781   \ifbbl@gregleap}
7782 \def\bbl@gregdayspriormonths#1#2#3{%
7783     {#3=\ifcase #1 0 \or 0 \or 31 \or 59 \or 90 \or 120 \or 151 \or
7784         181 \or 212 \or 243 \or 273 \or 304 \or 334 \fi
7785     \bbl@ifgregleap{#2}%
7786         \ifnum #1 > 2
7787             \advance #3 by 1
7788         \fi
7789     \fi
7790     \global\bbl@cntcommon=#3}%
7791     #3=\bbl@cntcommon}
7792 \def\bbl@gregdaysprioryears#1#2{%
7793   {\countdef\tmpc=4
7794   \countdef\tmpb=2
7795   \tmpb=#1\relax
7796   \advance \tmpb by -1
7797   \tmpc=\tmpb
7798   \multiply \tmpc by 365
7799   #2=\tmpc
7800   \tmpc=\tmpb
7801   \divide \tmpc by 4
7802   \advance #2 by \tmpc
7803   \tmpc=\tmpb
7804   \divide \tmpc by 100
7805   \advance #2 by -\tmpc
7806   \tmpc=\tmpb
7807   \divide \tmpc by 400
7808   \advance #2 by \tmpc
```

```
7809    \global\bbl@cntcommon=#2\relax}%
7810  #2=\bbl@cntcommon}
7811 \def\bbl@absfromgreg#1#2#3#4{%
7812  {\countdef\tmpd=0
7813   #4=#1\relax
7814   \bbl@gregdayspriormonths{#2}{#3}{\tmpd}%
7815   \advance #4 by \tmpd
7816   \bbl@gregdaysprioryears{#3}{\tmpd}%
7817   \advance #4 by \tmpd
7818   \global\bbl@cntcommon=#4\relax}%
7819  #4=\bbl@cntcommon}
7820 \newif\ifbbl@hebrleap
7821 \def\bbl@checkleaphebryear#1{%
7822  {\countdef\tmpa=0
7823   \countdef\tmpb=1
7824   \tmpa=#1\relax
7825   \multiply \tmpa by 7
7826   \advance \tmpa by 1
7827   \bbl@remainder{\tmpa}{19}{\tmpb}%
7828   \ifnum \tmpb < 7
7829       \global\bbl@hebrleaptrue
7830   \else
7831       \global\bbl@hebrleapfalse
7832   \fi}}
7833 \def\bbl@hebrelapsedmonths#1#2{%
7834  {\countdef\tmpa=0
7835   \countdef\tmpb=1
7836   \countdef\tmpc=2
7837   \tmpa=#1\relax
7838   \advance \tmpa by -1
7839   #2=\tmpa
7840   \divide #2 by 19
7841   \multiply #2 by 235
7842   \bbl@remainder{\tmpa}{19}{\tmpb}% \tmpa=years%19-years this cycle
7843   \tmpc=\tmpb
7844   \multiply \tmpb by 12
7845   \advance #2 by \tmpb
7846   \multiply \tmpc by 7
7847   \advance \tmpc by 1
7848   \divide \tmpc by 19
7849   \advance #2 by \tmpc
7850   \global\bbl@cntcommon=#2}%
7851  #2=\bbl@cntcommon}
7852 \def\bbl@hebrelapseddays#1#2{%
7853  {\countdef\tmpa=0
7854   \countdef\tmpb=1
7855   \countdef\tmpc=2
7856   \bbl@hebrelapsedmonths{#1}{#2}%
7857   \tmpa=#2\relax
7858   \multiply \tmpa by 13753
7859   \advance \tmpa by 5604
7860   \bbl@remainder{\tmpa}{25920}{\tmpc}% \tmpc == ConjunctionParts
7861   \divide \tmpa by 25920
7862   \multiply #2 by 29
7863   \advance #2 by 1
7864   \advance #2 by \tmpa
7865   \bbl@remainder{#2}{7}{\tmpa}%
7866   \ifnum \tmpc < 19440
7867       \ifnum \tmpc < 9924
7868       \else
7869           \ifnum \tmpa=2
7870               \bbl@checkleaphebryear{#1}% of a common year
7871               \ifbbl@hebrleap
```

214

```
7872            \else
7873                \advance #2 by 1
7874            \fi
7875          \fi
7876      \fi
7877      \ifnum \tmpc < 16789
7878      \else
7879          \ifnum \tmpa=1
7880              \advance #1 by -1
7881              \bbl@checkleaphebryear{#1}% at the end of leap year
7882              \ifbbl@hebrleap
7883                  \advance #2 by 1
7884              \fi
7885          \fi
7886      \fi
7887    \else
7888        \advance #2 by 1
7889    \fi
7890    \bbl@remainder{#2}{7}{\tmpa}%
7891    \ifnum \tmpa=0
7892        \advance #2 by 1
7893    \else
7894        \ifnum \tmpa=3
7895            \advance #2 by 1
7896        \else
7897            \ifnum \tmpa=5
7898                \advance #2 by 1
7899            \fi
7900        \fi
7901    \fi
7902    \global\bbl@cntcommon=#2\relax}%
7903  #2=\bbl@cntcommon}
7904 \def\bbl@daysinhebryear#1#2{%
7905  {\countdef\tmpe=12
7906    \bbl@hebrelapseddays{#1}{\tmpe}%
7907    \advance #1 by 1
7908    \bbl@hebrelapseddays{#1}{#2}%
7909    \advance #2 by -\tmpe
7910    \global\bbl@cntcommon=#2}%
7911  #2=\bbl@cntcommon}
7912 \def\bbl@hebrdayspriormonths#1#2#3{%
7913  {\countdef\tmpf= 14
7914    #3=\ifcase #1\relax
7915        0 \or
7916        0 \or
7917       30 \or
7918       59 \or
7919       89 \or
7920      118 \or
7921      148 \or
7922      148 \or
7923      177 \or
7924      207 \or
7925      236 \or
7926      266 \or
7927      295 \or
7928      325 \or
7929      400
7930    \fi
7931    \bbl@checkleaphebryear{#2}%
7932    \ifbbl@hebrleap
7933        \ifnum #1 > 6
7934            \advance #3 by 30
```

```
7935        \fi
7936      \fi
7937      \bbl@daysinhebryear{#2}{\tmpf}%
7938      \ifnum #1 > 3
7939          \ifnum \tmpf=353
7940              \advance #3 by -1
7941          \fi
7942          \ifnum \tmpf=383
7943              \advance #3 by -1
7944          \fi
7945      \fi
7946      \ifnum #1 > 2
7947          \ifnum \tmpf=355
7948              \advance #3 by 1
7949          \fi
7950          \ifnum \tmpf=385
7951              \advance #3 by 1
7952          \fi
7953      \fi
7954      \global\bbl@cntcommon=#3\relax}%
7955    #3=\bbl@cntcommon}
7956 \def\bbl@absfromhebr#1#2#3#4{%
7957    {#4=#1\relax
7958      \bbl@hebrdayspriormonths{#2}{#3}{#1}%
7959      \advance #4 by #1\relax
7960      \bbl@hebrelapseddays{#3}{#1}%
7961      \advance #4 by #1\relax
7962      \advance #4 by -1373429
7963      \global\bbl@cntcommon=#4\relax}%
7964    #4=\bbl@cntcommon}
7965 \def\bbl@hebrfromgreg#1#2#3#4#5#6{%
7966    {\countdef\tmpx= 17
7967      \countdef\tmpy= 18
7968      \countdef\tmpz= 19
7969      #6=#3\relax
7970      \global\advance #6 by 3761
7971      \bbl@absfromgreg{#1}{#2}{#3}{#4}%
7972      \tmpz=1   \tmpy=1
7973      \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
7974      \ifnum \tmpx > #4\relax
7975          \global\advance #6 by -1
7976          \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
7977      \fi
7978      \advance #4 by -\tmpx
7979      \advance #4 by 1
7980      #5=#4\relax
7981      \divide #5 by 30
7982      \loop
7983          \bbl@hebrdayspriormonths{#5}{#6}{\tmpx}%
7984          \ifnum \tmpx < #4\relax
7985              \advance #5 by 1
7986              \tmpy=\tmpx
7987      \repeat
7988      \global\advance #5 by -1
7989      \global\advance #4 by -\tmpy}}
7990 \newcount\bbl@hebrday \newcount\bbl@hebrmonth \newcount\bbl@hebryear
7991 \newcount\bbl@gregday \newcount\bbl@gregmonth \newcount\bbl@gregyear
7992 \def\bbl@ca@hebrew#1-#2-#3\@@#4#5#6{%
7993    \bbl@gregday=#3\relax \bbl@gregmonth=#2\relax \bbl@gregyear=#1\relax
7994    \bbl@hebrfromgreg
7995      {\bbl@gregday}{\bbl@gregmonth}{\bbl@gregyear}%
7996      {\bbl@hebrday}{\bbl@hebrmonth}{\bbl@hebryear}%
7997    \edef#4{\the\bbl@hebryear}%
```

```
7998    \edef#5{\the\bbl@hebrmonth}%
7999    \edef#6{\the\bbl@hebrday}}
8000 ⟨/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).

```
8001 ⟨∗ca-persian⟩
8002 \ExplSyntaxOn
8003 ⟨⟨Compute Julian day⟩⟩
8004 \def\bbl@cs@firstjal@xx{2012,2016,2020,2024,2028,2029,% March 20
8005    2032,2033,2036,2037,2040,2041,2044,2045,2048,2049}
8006 \def\bbl@ca@persian#1-#2-#3\@@#4#5#6{%
8007    \edef\bbl@tempa{#1}%  20XX-03-\bbl@tempe = 1 farvardin:
8008    \ifnum\bbl@tempa>2012 \ifnum\bbl@tempa<2051
8009      \bbl@afterfi\expandafter\@gobble
8010    \fi\fi
8011    {\bbl@error{Year~out~of~range}{The~allowed~range~is~2013-2050}}%
8012    \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
8013    \ifin@\def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
8014    \edef\bbl@tempc{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{#2}{#3}+.5}}% current
8015    \edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}% begin
8016    \ifnum\bbl@tempc<\bbl@tempb
8017      \edef\bbl@tempa{\fp_eval:n{\bbl@tempa-1}}% go back 1 year and redo
8018      \bbl@xin@{\bbl@tempa}{\bbl@cs@firstjal@xx}%
8019      \ifin@\def\bbl@tempe{20}\else\def\bbl@tempe{21}\fi
8020      \edef\bbl@tempb{\fp_eval:n{\bbl@cs@jd{\bbl@tempa}{03}{\bbl@tempe}+.5}}%
8021    \fi
8022    \edef#4{\fp_eval:n{\bbl@tempa-621}}% set Jalali year
8023    \edef#6{\fp_eval:n{\bbl@tempc-\bbl@tempb+1}}% days from 1 farvardin
8024    \edef#5{\fp_eval:n{% set Jalali month
8025      (#6 <= 186) ? ceil(#6 / 31) : ceil((#6 - 6) / 30)}}
8026    \edef#6{\fp_eval:n{% set Jalali day
8027      (#6 - ((#5 <= 7) ? ((#5 - 1) * 31) : (((#5 - 1) * 30) + 6)))}}}}
8028 \ExplSyntaxOff
8029 ⟨/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.

```
8030 ⟨∗ca-coptic⟩
8031 \ExplSyntaxOn
8032 ⟨⟨Compute Julian day⟩⟩
8033 \def\bbl@ca@coptic#1-#2-#3\@@#4#5#6{%
8034    \edef\bbl@tempd{\fp_eval:n{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
8035    \edef\bbl@tempc{\fp_eval:n{\bbl@tempd - 1825029.5}}%
8036    \edef#4{\fp_eval:n{%
8037      floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461) / 365) + 1}}%
8038    \edef\bbl@tempc{\fp_eval:n{%
8039      \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1825029.5}}%
8040    \edef#5{\fp_eval:n{floor(\bbl@tempc / 30) + 1}}%
8041    \edef#6{\fp_eval:n{\bbl@tempc - (#5 - 1) * 30 + 1}}}
8042 \ExplSyntaxOff
8043 ⟨/ca-coptic⟩
8044 ⟨∗ca-ethiopic⟩
8045 \ExplSyntaxOn
```

8046 ⟨⟨*Compute Julian day*⟩⟩
8047 \def\bbl@ca@ethiopic#1-#2-#3\@@#4#5#6{%
8048   \edef\bbl@tempd{\fp_eval:n{floor(\bbl@cs@jd{#1}{#2}{#3}) + 0.5}}%
8049   \edef\bbl@tempc{\fp_eval:n{\bbl@tempd - 1724220.5}}%
8050   \edef#4{\fp_eval:n{%
8051   floor((\bbl@tempc - floor((\bbl@tempc+366) / 1461)) / 365) + 1}}%
8052   \edef\bbl@tempc{\fp_eval:n{%
8053   \bbl@tempd - (#4-1) * 365 - floor(#4/4) - 1724220.5}}%
8054   \edef#5{\fp_eval:n{floor(\bbl@tempc / 30) + 1}}%
8055   \edef#6{\fp_eval:n{\bbl@tempc - (#5 - 1) * 30 + 1}}}
8056 \ExplSyntaxOff
8057 ⟨/ca-ethiopic⟩

19   Buddhist

That's very simple.

8058 ⟨*ca-buddhist⟩
8059 \def\bbl@ca@buddhist#1-#2-#3\@@#4#5#6{%
8060   \edef#4{\number\numexpr#1+543\relax}%
8061   \edef#5{#2}%
8062   \edef#6{#3}}
8063 ⟨/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.

8064 ⟨*bplain | blplain⟩
8065 \catcode`\{=1 % left brace is begin-group character
8066 \catcode`\}=2 % right brace is end-group character
8067 \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).

8068 \openin 0 hyphen.cfg
8069 \ifeof0
8070 \else
8071   \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.

8072   \def\input #1 {%
8073   \let\input\a
8074   \a hyphen.cfg
8075   \let\a\undefined

```
8076    }
8077 \fi
8078 ⟨/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.

```
8079 ⟨bplain⟩\a plain.tex
8080 ⟨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.

```
8081 ⟨bplain⟩\def\fmtname{babel-plain}
8082 ⟨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).

```
8083 ⟨⟨*Emulate LaTeX⟩⟩ ≡
8084 \def\@empty{}
8085 \def\loadlocalcfg#1{%
8086   \openin0#1.cfg
8087   \ifeof0
8088     \closein0
8089   \else
8090     \closein0
8091     {\immediate\write16{************************************}%
8092      \immediate\write16{* Local config file #1.cfg used}%
8093      \immediate\write16{*}%
8094      }
8095     \input #1.cfg\relax
8096   \fi
8097 \@endofldf}
```

20.3 General tools

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

```
8098 \long\def\@firstofone#1{#1}
8099 \long\def\@firstoftwo#1#2{#1}
8100 \long\def\@secondoftwo#1#2{#2}
8101 \def\@nnil{\@nil}
8102 \def\@gobbletwo#1#2{}
8103 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
8104 \def\@star@or@long#1{%
8105   \@ifstar
8106   {\let\l@ngrel@x\relax#1}%
8107   {\let\l@ngrel@x\long#1}}
8108 \let\l@ngrel@x\relax
8109 \def\@car#1#2\@nil{#1}
8110 \def\@cdr#1#2\@nil{#2}
8111 \let\@typeset@protect\relax
8112 \let\protected@edef\edef
8113 \long\def\@gobble#1{}
8114 \edef\@backslashchar{\expandafter\@gobble\string\\}
8115 \def\strip@prefix#1>{}
8116 \def\g@addto@macro#1#2{{%
8117     \toks@\expandafter{#1#2}%
```

```
8118        \xdef#1{\the\toks@}}}
8119 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
8120 \def\@nameuse#1{\csname #1\endcsname}
8121 \def\@ifundefined#1{%
8122   \expandafter\ifx\csname#1\endcsname\relax
8123     \expandafter\@firstoftwo
8124   \else
8125     \expandafter\@secondoftwo
8126   \fi}
8127 \def\@expandtwoargs#1#2#3{%
8128   \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
8129 \def\zap@space#1 #2{%
8130   #1%
8131   \ifx#2\@empty\else\expandafter\zap@space\fi
8132   #2}
8133 \let\bbl@trace\@gobble
8134 \def\bbl@error#1#2{%
8135   \begingroup
8136     \newlinechar=`\^^J
8137     \def\\{^^J(babel) }%
8138     \errhelp{#2}\errmessage{\\#1}%
8139   \endgroup}
8140 \def\bbl@warning#1{%
8141   \begingroup
8142     \newlinechar=`\^^J
8143     \def\\{^^J(babel) }%
8144     \message{\\#1}%
8145   \endgroup}
8146 \let\bbl@infowarn\bbl@warning
8147 \def\bbl@info#1{%
8148   \begingroup
8149     \newlinechar=`\^^J
8150     \def\\{^^J}%
8151     \wlog{#1}%
8152   \endgroup}
```

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

```
8153 \ifx\@preamblecmds\@undefined
8154   \def\@preamblecmds{}
8155 \fi
8156 \def\@onlypreamble#1{%
8157   \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
8158     \@preamblecmds\do#1}}
8159 \@onlypreamble\@onlypreamble
```

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

```
8160 \def\begindocument{%
8161   \@begindocumenthook
8162   \global\let\@begindocumenthook\@undefined
8163   \def\do##1{\global\let##1\@undefined}%
8164   \@preamblecmds
8165   \global\let\do\noexpand}
8166 \ifx\@begindocumenthook\@undefined
8167   \def\@begindocumenthook{}
8168 \fi
8169 \@onlypreamble\@begindocumenthook
8170 \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.

```
8171 \def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
8172 \@onlypreamble\AtEndOfPackage
```

```
8173 \def\@endofldf{}
8174 \@onlypreamble\@endofldf
8175 \let\bbl@afterlang\@empty
8176 \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.

```
8177 \catcode`\&=\z@
8178 \ifx&if@filesw\@undefined
8179    \expandafter\let\csname if@filesw\expandafter\endcsname
8180       \csname iffalse\endcsname
8181 \fi
8182 \catcode`\&=4
```

Mimick LaTeX's commands to define control sequences.

```
8183 \def\newcommand{\@star@or@long\new@command}
8184 \def\new@command#1{%
8185    \@testopt{\@newcommand#1}0}
8186 \def\@newcommand#1[#2]{%
8187    \@ifnextchar [{\@xargdef#1[#2]}%
8188                  {\@argdef#1[#2]}}
8189 \long\def\@argdef#1[#2]#3{%
8190    \@yargdef#1\@ne{#2}{#3}}
8191 \long\def\@xargdef#1[#2][#3]#4{%
8192    \expandafter\def\expandafter#1\expandafter{%
8193       \expandafter\@protected@testopt\expandafter #1%
8194       \csname\string#1\expandafter\endcsname{#3}}%
8195    \expandafter\@yargdef \csname\string#1\endcsname
8196    \tw@{#2}{#4}}
8197 \long\def\@yargdef#1#2#3{%
8198    \@tempcnta#3\relax
8199    \advance \@tempcnta \@ne
8200    \let\@hash@\relax
8201    \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
8202    \@tempcntb #2%
8203    \@whilenum\@tempcntb <\@tempcnta
8204    \do{%
8205       \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
8206       \advance\@tempcntb \@ne}%
8207    \let\@hash@##%
8208    \l@ngrel@x\expandafter\def\expandafter#1\reserved@a}
8209 \def\providecommand{\@star@or@long\provide@command}
8210 \def\provide@command#1{%
8211    \begingroup
8212       \escapechar\m@ne\xdef\@gtempa{{\string#1}}%
8213    \endgroup
8214    \expandafter\@ifundefined\@gtempa
8215       {\def\reserved@a{\new@command#1}}%
8216       {\let\reserved@a\relax
8217        \def\reserved@a{\new@command\reserved@a}}%
8218    \reserved@a}%
8219 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
8220 \def\declare@robustcommand#1{%
8221    \edef\reserved@a{\string#1}%
8222    \def\reserved@b{#1}%
8223    \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
8224    \edef#1{%
8225       \ifx\reserved@a\reserved@b
8226          \noexpand\x@protect
8227          \noexpand#1%
8228       \fi
8229       \noexpand\protect
```

```
8230        \expandafter\noexpand\csname
8231            \expandafter\@gobble\string#1 \endcsname
8232    }%
8233    \expandafter\new@command\csname
8234        \expandafter\@gobble\string#1 \endcsname
8235 }
8236 \def\x@protect#1{%
8237    \ifx\protect\@typeset@protect\else
8238        \@x@protect#1%
8239    \fi
8240 }
8241 \catcode`\&=\z@  % Trick to hide conditionals
8242    \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.

```
8243    \def\bbl@tempa{\csname newif\endcsname&ifin@}
8244 \catcode`\&=4
8245 \ifx\in@\@undefined
8246   \def\in@#1#2{%
8247     \def\in@@##1#1##2##3\in@@{%
8248       \ifx\in@##2\in@false\else\in@true\fi}%
8249     \in@@#2#1\in@\in@@}
8250 \else
8251   \let\bbl@tempa\@empty
8252 \fi
8253 \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).

```
8254 \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.

```
8255 \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.

```
8256 \ifx\@tempcnta\@undefined
8257   \csname newcount\endcsname\@tempcnta\relax
8258 \fi
8259 \ifx\@tempcntb\@undefined
8260   \csname newcount\endcsname\@tempcntb\relax
8261 \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).

```
8262 \ifx\bye\@undefined
8263   \advance\count10 by -2\relax
8264 \fi
8265 \ifx\@ifnextchar\@undefined
8266   \def\@ifnextchar#1#2#3{%
8267     \let\reserved@d=#1%
8268     \def\reserved@a{#2}\def\reserved@b{#3}%
8269     \futurelet\@let@token\@ifnch}
8270   \def\@ifnch{%
8271     \ifx\@let@token\@sptoken
8272       \let\reserved@c\@xifnch
8273     \else
```

```
8274        \ifx\@let@token\reserved@d
8275          \let\reserved@c\reserved@a
8276        \else
8277          \let\reserved@c\reserved@b
8278        \fi
8279     \fi
8280     \reserved@c}
8281   \def\:{\let\@sptoken= } \:  % this makes \@sptoken a space token
8282   \def\:{\@xifnch} \expandafter\def\: {\futurelet\@let@token\@ifnch}
8283 \fi
8284 \def\@testopt#1#2{%
8285   \@ifnextchar[{#1}{#1[#2]}}
8286 \def\@protected@testopt#1{%
8287   \ifx\protect\@typeset@protect
8288     \expandafter\@testopt
8289   \else
8290     \@x@protect#1%
8291   \fi}
8292 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
8293       #2\relax}\fi}
8294 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
8295         \else\expandafter\@gobble\fi{#1}}
```

20.4   Encoding related macros

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

```
8296 \def\DeclareTextCommand{%
8297     \@dec@text@cmd\providecommand
8298 }
8299 \def\ProvideTextCommand{%
8300     \@dec@text@cmd\providecommand
8301 }
8302 \def\DeclareTextSymbol#1#2#3{%
8303     \@dec@text@cmd\chardef#1{#2}#3\relax
8304 }
8305 \def\@dec@text@cmd#1#2#3{%
8306     \expandafter\def\expandafter#2%
8307       \expandafter{%
8308         \csname#3-cmd\expandafter\endcsname
8309         \expandafter#2%
8310         \csname#3\string#2\endcsname
8311       }%
8312 %   \let\@ifdefinable\@rc@ifdefinable
8313     \expandafter#1\csname#3\string#2\endcsname
8314 }
8315 \def\@current@cmd#1{%
8316   \ifx\protect\@typeset@protect\else
8317       \noexpand#1\expandafter\@gobble
8318   \fi
8319 }
8320 \def\@changed@cmd#1#2{%
8321     \ifx\protect\@typeset@protect
8322       \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
8323         \expandafter\ifx\csname ?\string#1\endcsname\relax
8324           \expandafter\def\csname ?\string#1\endcsname{%
8325             \@changed@x@err{#1}%
8326           }%
8327         \fi
8328         \global\expandafter\let
8329           \csname\cf@encoding \string#1\expandafter\endcsname
8330           \csname ?\string#1\endcsname
8331       \fi
8332       \csname\cf@encoding\string#1%
```

```
8333        \expandafter\endcsname
8334    \else
8335        \noexpand#1%
8336    \fi
8337 }
8338 \def\@changed@x@err#1{%
8339    \errhelp{Your command will be ignored, type <return> to proceed}%
8340    \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
8341 \def\DeclareTextCommandDefault#1{%
8342    \DeclareTextCommand#1?%
8343 }
8344 \def\ProvideTextCommandDefault#1{%
8345    \ProvideTextCommand#1?%
8346 }
8347 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
8348 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
8349 \def\DeclareTextAccent#1#2#3{%
8350   \DeclareTextCommand#1{#2}[1]{\accent#3 ##1}
8351 }
8352 \def\DeclareTextCompositeCommand#1#2#3#4{%
8353    \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
8354    \edef\reserved@b{\string##1}%
8355    \edef\reserved@c{%
8356      \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
8357    \ifx\reserved@b\reserved@c
8358      \expandafter\expandafter\expandafter\ifx
8359        \expandafter\@car\reserved@a\relax\relax\@nil
8360        \@text@composite
8361      \else
8362        \edef\reserved@b##1{%
8363          \def\expandafter\noexpand
8364            \csname#2\string#1\endcsname####1{%
8365            \noexpand\@text@composite
8366              \expandafter\noexpand\csname#2\string#1\endcsname
8367              ####1\noexpand\@empty\noexpand\@text@composite
8368              {##1}%
8369          }%
8370        }%
8371        \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
8372      \fi
8373      \expandafter\def\csname\expandafter\string\csname
8374          #2\endcsname\string#1-\string#3\endcsname{#4}
8375    \else
8376      \errhelp{Your command will be ignored, type <return> to proceed}%
8377      \errmessage{\string\DeclareTextCompositeCommand\space used on
8378          inappropriate command \protect#1}
8379    \fi
8380 }
8381 \def\@text@composite#1#2#3\@text@composite{%
8382    \expandafter\@text@composite@x
8383      \csname\string#1-\string#2\endcsname
8384 }
8385 \def\@text@composite@x#1#2{%
8386    \ifx#1\relax
8387        #2%
8388    \else
8389        #1%
8390    \fi
8391 }
8392 %
8393 \def\@strip@args#1:#2-#3\@strip@args{#2}
8394 \def\DeclareTextComposite#1#2#3#4{%
8395    \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
```

224

```
8396     \bgroup
8397       \lccode`\@=#4%
8398       \lowercase{%
8399     \egroup
8400       \reserved@a @%
8401     }%
8402 }
8403 %
8404 \def\UseTextSymbol#1#2{#2}
8405 \def\UseTextAccent#1#2#3{}
8406 \def\@use@text@encoding#1{}
8407 \def\DeclareTextSymbolDefault#1#2{%
8408     \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
8409 }
8410 \def\DeclareTextAccentDefault#1#2{%
8411     \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
8412 }
8413 \def\cf@encoding{OT1}
```

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

```
8414 \DeclareTextAccent{\"}{OT1}{127}
8415 \DeclareTextAccent{\'}{OT1}{19}
8416 \DeclareTextAccent{\^}{OT1}{94}
8417 \DeclareTextAccent{\`}{OT1}{18}
8418 \DeclareTextAccent{\~}{OT1}{126}
```

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

```
8419 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
8420 \DeclareTextSymbol{\textquotedblright}{OT1}{`\"}
8421 \DeclareTextSymbol{\textquoteleft}{OT1}{`\`}
8422 \DeclareTextSymbol{\textquoteright}{OT1}{`\'}
8423 \DeclareTextSymbol{\i}{OT1}{16}
8424 \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.

```
8425 \ifx\scriptsize\@undefined
8426   \let\scriptsize\sevenrm
8427 \fi
```

And a few more "dummy" definitions.

```
8428 \def\languagename{english}%
8429 \let\bbl@opt@shorthands\@nnil
8430 \def\bbl@ifshorthand#1#2#3{#2}%
8431 \let\bbl@language@opts\@empty
8432 \ifx\babeloptionstrings\@undefined
8433   \let\bbl@opt@strings\@nnil
8434 \else
8435   \let\bbl@opt@strings\babeloptionstrings
8436 \fi
8437 \def\BabelStringsDefault{generic}
8438 \def\bbl@tempa{normal}
8439 \ifx\babeloptionmath\bbl@tempa
8440   \def\bbl@mathnormal{\noexpand\textormath}
8441 \fi
8442 \def\AfterBabelLanguage#1#2{}
8443 \ifx\BabelModifiers\@undefined\let\BabelModifiers\relax\fi
8444 \let\bbl@afterlang\relax
8445 \def\bbl@opt@safe{BR}
8446 \ifx\@uclclist\@undefined\let\@uclclist\@empty\fi
8447 \ifx\bbl@trace\@undefined\def\bbl@trace#1{}\fi
8448 \expandafter\newif\csname ifbbl@single\endcsname
8449 \chardef\bbl@bidimode\z@
8450 ⟨⟨/Emulate LaTeX⟩⟩
```

A proxy file:

```
8451 ⟨*plain⟩
8452 \input babel.def
8453 ⟨/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).