Babel

Javier Bezos
Current maintainer

Johannes L. Braams
Original author

Localization and
internationalization

Unicode
T_EX
pdfT_EX
LuaT_EX
XeT_EX

Contents

1

Troubleshoooting

Part I
User guide

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

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

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

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

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

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

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

1 The user interface

1.1 Monolingual documents

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

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

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

```
PDFTEX

    \documentclass{article}

    \usepackage[T1]{fontenc}
```

```
\usepackage[french]{babel}

\begin{document}

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

\end{document}
```

Now consider something like:

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

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

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

LUATEX/XETEX

```
\documentclass[russian]{article}

\usepackage{babel}

\babelfont{rm}{DejaVu Serif}

\begin{document}

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

\end{document}
```

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

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

Or the more explanatory:

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

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

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

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

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

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

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

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

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

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

1.2 Multilingual documents

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

EXAMPLE In LaTeX, the preamble of the document:

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

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

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

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

Examples of cases where main is useful are the following.

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

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

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

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

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

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

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

```
PDFTEX
\documentclass{article}

\usepackage[T1]{fontenc}

\usepackage[english,french]{babel}

\begin{document}

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

\selectlanguage{english}

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

\end{document}
```

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

```
LUATEX/XETEX
\documentclass{article}

\usepackage[vietnamese,danish]{babel}

\begin{document}

\prefacename{} -- \alsoname{} -- \today

\selectlanguage{vietnamese}

\prefacename{} -- \alsoname{} -- \today

\end{document}
```

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

1.3 Mostly monolingual documents

New 3.39 Very often, multilingual documents consist of a main language with small pieces of text in another languages (words, idioms, short sentences). Typically, all you need is to set the line breaking rules and, perhaps, the font. In such a case, babel now does not require declaring these secondary languages explicitly, because the basic settings are loaded on the fly when the language is selected (and also when provided in the optional argument of `\babelfont`, if used.)
This is particularly useful, too, when there are short texts of this kind coming from an external source whose contents are not known on beforehand (for example, titles in a bibliography). At this regard, it is worth remembering that `\babelfont` does *not* load any font until required, so that it can be used just in case.

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

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

\babelfont[russian]{rm}{FreeSerif}

\begin{document}

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

\end{document}
```

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

1.4 Modifiers

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

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

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

1.5 Troubleshooting

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

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

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

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

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

---

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

1.6 Plain

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

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

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

1.7 Basic language selectors

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

\selectlanguage   {⟨*language*⟩}

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

```
\selectlanguage{german}
```

This command can be used as environment, too.

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

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

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

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

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

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

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

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

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

The command \foreignlanguage takes two arguments; the second argument is a phrase to be typeset according to the rules of the language named in its first one.
This command (1) only switches the extra definitions and the hyphenation rules for the language, *not* the names and dates, (2) does not send information about the language to auxiliary files (i.e., the surrounding language is still in force), and (3) it works even if the language has not been set as package option (but in such a case it only sets the hyphenation patterns and a warning is shown). With the bidi option, it also enters in horizontal mode (this is not done always for backwards compatibility), and since it is meant for phrases only the text direction (and not the paragraph one) is set.
New 3.44 As already said, captions and dates are not switched. However, with the optional argument you can switch them, too. So, you can write:

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

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

1.8   Auxiliary language selectors

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

The environment otherlanguage does basically the same as \selectlanguage, except that language change is (mostly) local to the environment.
Actually, there might be some non-local changes, as this environment is roughly equivalent to:

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

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

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

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

1.9   More on selection

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

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

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

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

EXAMPLE With

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

you can write

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

and

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

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

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

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

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

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

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

```
\babelensure{polish}
```

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

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

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

---

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

1.10 Shorthands

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

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

NOTE Keep in mind the following:

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

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

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

TROUBLESHOOTING A typical error when using shorthands is the following:

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

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

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

It is sometimes necessary to switch a shorthand character off temporarily, because it must be used in an entirely different way. For this purpose, the user commands \shorthandoff and \shorthandon are provided. They each take a list of characters as their arguments. The command \shorthandoff sets the \catcode for each of the characters in its argument to other (12); the command \shorthandon sets the \catcode to active (13). Both commands only work on 'known' shorthand characters.

New 3.9a However, \shorthandoff does not behave as you would expect with characters like ~ or ^, because they usually are not "other". For them \shorthandoff* is provided, so that with

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

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

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

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

`\useshorthands`   `*{⟨char⟩}`

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

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

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

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

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

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

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

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

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

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

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

`\languageshorthands`   `{⟨language⟩}`

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

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

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

---

[5]Actually, any name not corresponding to a language group does the same as none. However, follow this convention because it might be enforced in future releases of babel to catch possible errors.

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

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

\babelshorthand {⟨*shorthand*⟩}

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

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

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

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

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

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

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

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

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

The command `\aliasshorthand` can be used to let another character perform the same functions as the default shorthand character. If one prefers for example to use the

---

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

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

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

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

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

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

1.11   Package options

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

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

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

activegrave   Same for `.

shorthands=   ⟨*char*⟩⟨*char*⟩... | off

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

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

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

safe=   none | ref | bib

Some LaTeX macros are redefined so that using shorthands is safe. With safe=bib only
\nocite, \bibcite and \bibitem are redefined. With safe=ref only \newlabel, \ref and
\pageref are redefined (as well as a few macros from varioref and ifthen).
With safe=none no macro is redefined. This option is strongly recommended, because a
good deal of incompatibilities and errors are related to these redefinitions. As of
New 3.34   , in ϵTeX based engines (ie, almost every engine except the oldest ones)
shorthands can be used in these macros (formerly you could not).

math=   active | normal

Shorthands are mainly intended for text, not for math. By setting this option with the
value normal they are deactivated in math mode (default is active) and things like ${a'}$
(a closing brace after a shorthand) are not a source of trouble anymore.

config=   ⟨*file*⟩

Load ⟨*file*⟩`.cfg` instead of the default config file `bblopts.cfg` (the file is loaded even with `noconfigs`).

main=   ⟨*language*⟩

Sets the main language, as explained above, ie, this language is always loaded last. If it is not given as package or global option, it is added to the list of requested languages.

headfoot=   ⟨*language*⟩

By default, headlines and footlines are not touched (only marks), and if they contain language-dependent macros (which is not usual) there may be unexpected results. With this option you may set the language in heads and foots.

noconfigs   Global and language default config files are not loaded, so you can make sure your document is not spoilt by an unexpected `.cfg` file. However, if the key `config` is set, this file is loaded.

showlanguages   Prints to the log the list of languages loaded when the format was created: number (remember dialects can share it), name, hyphenation file and exceptions file.

nocase   New 3.9l   Language settings for uppercase and lowercase mapping (as set by `\SetCase`) are ignored. Use only if there are incompatibilities with other packages.

silent   New 3.9l   No warnings and no *infos* are written to the log file.[8]

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

Selects the encoding of strings in languages supporting this feature. Predefined labels are `generic` (for traditional TeX, LICR and ASCII strings), `unicode` (for engines like xetex and luatex) and `encoded` (for special cases requiring mixed encodings). Other allowed values are font encoding codes (T1, T2A, LGR, L7X...), but only in languages supporting them. Be aware with encoded captions are protected, but they work in `\MakeUppercase` and the like (this feature misuses some internal LaTeX tools, so use it only as a last resort).

hyphenmap=   `off | first | select | other | other*`

New 3.9g   Sets the behavior of case mapping for hyphenation, provided the language defines it.[9] It can take the following values:

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

---

[8]You can use alternatively the package silence.

[9]Turned off in plain.

[10]Duplicated options count as several ones.

[11]Providing `foreign` is pointless, because the case mapping applied is that at the end of the paragraph, but if either xetex or luatex change this behavior it might be added. On the other hand, `other` is provided even if I [JBL] think it isn't really useful, but who knows.

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

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

layout=	

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

provide=	*

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

1.12 The base option

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

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

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

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

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

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

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

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

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

1.13 ini files

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

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

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

LUATEX/XETEX

```
\documentclass{book}

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

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

\begin{document}

\tableofcontents

\chapter{სამზარეულო და სუფრის ტრადიციები}

ქართული ტრადიციული სამზარეულო ერთ-ერთი უმდიდრესია მთელ მსოფლიოში.

\end{document}
```

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

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

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

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

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

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

Or also:

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

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

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

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

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

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

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

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

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

```
\babelprovide[import, hyphenrules=+]{lao}
\babelpatterns[lao]{1ກ 1ຢ 1ຣ 1ງ 1ກ 1ໆ} % Random
```

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

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

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

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

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

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

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

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

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

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

---

aghem
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
bengali
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

---

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

french-luxembourg
french-switzerland
french
friulian
fulah
galician
ganda
georgian
german-at
german-austria
german-ch
german-switzerland
german
greek
gujarati
gusii
hausa-gh
hausa-ghana
hausa-ne
hausa-niger
hausa
hawaiian
hebrew
hindi
hungarian
icelandic
igbo
inarisami
indonesian
interlingua
irish
italian
japanese
jolafonyi
kabuverdianu
kabyle
kako
kalaallisut
kalenjin
kamba
kannada
kashmiri
kazakh
khmer
kikuyu
kinyarwanda
konkani
korean
koyraborosenni
koyrachiini
kwasio
kyrgyz
lakota
langi
lao
latvian
lingala
lithuanian

lowersorbian
lsorbian
lubakatanga
luo
luxembourgish
luyia
macedonian
machame
makhuwameetto
makonde
malagasy
malay-bn
malay-brunei
malay-sg
malay-singapore
malay
malayalam
maltese
manx
marathi
masai
mazanderani
meru
meta
mexican
mongolian
morisyen
mundang
nama
nepali
newzealand
ngiemboon
ngomba
norsk
northernluri
northernsami
northndebele
norwegianbokmal
norwegiannynorsk
nswissgerman
nuer
nyankole
nynorsk
occitan
oriya
oromo
ossetic
pashto
persian
piedmontese
polish
polytonicgreek
portuguese-br
portuguese-brazil
portuguese-portugal
portuguese-pt
portuguese
punjabi-arab

punjabi-arabic
punjabi-gurmukhi
punjabi-guru
punjabi
quechua
romanian
romansh
rombo
rundi
russian
rwa
sakha
samburu
samin
sango
sangu
sanskrit-beng
sanskrit-bengali
sanskrit-deva
sanskrit-devanagari
sanskrit-gujarati
sanskrit-gujr
sanskrit-kannada
sanskrit-knda
sanskrit-malayalam
sanskrit-mlym
sanskrit-telu
sanskrit-telugu
sanskrit
scottishgaelic
sena
serbian-cyrillic-bosniaherzegovina
serbian-cyrillic-kosovo
serbian-cyrillic-montenegro
serbian-cyrillic
serbian-cyrl-ba
serbian-cyrl-me
serbian-cyrl-xk
serbian-cyrl
serbian-latin-bosniaherzegovina
serbian-latin-kosovo
serbian-latin-montenegro
serbian-latin
serbian-latn-ba
serbian-latn-me
serbian-latn-xk
serbian-latn
serbian
shambala
shona
sichuanyi
sinhala
slovak
slovene
slovenian

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

Modifying and adding values to `ini` files

New 3.39   There is a way to modify the values of `ini` files when they get loaded with

\babelprovide and import. To set, say, digits.native in the numbers section, use
something like numbers/digits.native=abcdefghij. Keys may be added, too. Without
import you may modify the identification keys.

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

1.14   Selecting fonts

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

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

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

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

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

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

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

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

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

LUATEX/XETEX

```
\documentclass{article}

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

\babelprovide[import]{hebrew}

\babelfont{rm}{FreeSerif}

\begin{document}

Svenska \foreignlanguage{hebrew}{עִבְרִית} svenska.

\end{document}
```

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

LUATEX/XETEX

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

---

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

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

EXAMPLE Here is how to do it:

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

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

NOTE You may load fontspec explicitly. For example:

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

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

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

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

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

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

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

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

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

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

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

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

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

1.15 Modifying a language

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

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

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

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

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

NOTE There are a few alternative methods:

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

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

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

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

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

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

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

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

This redefinition is immediate.

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

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

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

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

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

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

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

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

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

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

1.16 Creating a language

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

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

If the language ⟨*language-name*⟩ has not been loaded as class or package option and there are no ⟨*options*⟩, it creates an "empty" one with some defaults in its internal structure: the hyphen rules, if not available, are set to the current ones, left and right hyphen mins are set to 2 and 3. In either case, caption, date and language system are not defined.

If no `ini` file is imported with `import`, ⟨*language-name*⟩ is still relevant because in such a case the hyphenation and like breaking rules (including those for South East Asian and CJK) are based on it as provided in the `ini` file corresponding to that name; the same applies to OpenType language and script.

Conveniently, some options allow to fill the language, and babel warns you about what to do if there is a missing string. Very likely you will find alerts like that in the `log` file:

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

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

EXAMPLE If you need a language named `arhinish`:

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

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

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

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

The main language is not changed (`danish` in this example). So, you must add `\selectlanguage{arhinish}` or other selectors where necessary.

If the language has been loaded as an argument in `\documentclass` or `\usepackage`, then `\babelprovide` redefines the requested data.

import= ⟨*language-tag*⟩

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

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

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

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

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

captions= ⟨*language-tag*⟩

Loads only the strings. For example:

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

hyphenrules= ⟨*language-list*⟩

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

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

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

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

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

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

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

script= ⟨*script-name*⟩

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

language= ⟨*language-name*⟩

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

alph= ⟨*counter-name*⟩

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

Alph= ⟨*counter-name*⟩

Same for `\Alph`.

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

onchar= `ids` | `fonts`

New 3.38 This option is much like an 'event' called when a character belonging to the script of this locale is found (as its name implies, it acts on characters, not on spaces). There are currently two 'actions', which can be used at the same time (separated by a space): with `ids` the `\language` and the `\localeid` are set to the values of this locale; with `fonts`, the fonts are changed to those of this locale (as set with `\babelfont`). This option is not compatible with mapfont. Characters can be added or modified with `\babelcharproperty`.

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

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

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

| intrapenalty= | ⟨*penalty*⟩ |

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

| transforms= | ⟨*transform-list*⟩ |

See section 1.21.

| justification= | kashida | elongated | unhyphenated |

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

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

| mapfont= | direction |

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

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

1.17   Digits and counters

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

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

Languages providing native digits in all or some variants are:

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

New 3.30 With luatex there is an alternative approach for mapping digits, namely, `mapdigits`. Conversion is based on the language and it is applied to the typeset text (not math, PDF bookmarks, etc.) before bidi and fonts are processed (ie, to the node list as generated by the TeX code). This means the local digits have the correct bidirectional behavior (unlike Numbers=Arabic in fontspec, which is not recommended).

NOTE With xetex you can use the option Mapping when defining a font.

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

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

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

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

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

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

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

The styles are:

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

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

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

1.18   Dates

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

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

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

1.19   Accessing language info

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

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

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

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

\localeinfo   * {⟨*field*⟩}

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

name.english as provided by the Unicode CLDR.
tag.ini is the tag of the ini file (the way this file is identified in its name).
tag.bcp47 is the full BCP 47 tag (see the warning below). This is the value to be used for the 'real' provided tag (babel may fill other fields if they are considered necessary).
language.tag.bcp47 is the BCP 47 language tag.
tag.opentype is the tag used by OpenType (usually, but not always, the same as BCP 47).
script.name , as provided by the Unicode CLDR.

script.tag.bcp47 is the BCP 47 tag of the script used by this locale. This is a required field for the fonts to be correctly set up, and therefore it should be always defined.

script.tag.opentype is the tag used by OpenType (usually, but not always, the same as BCP 47).

region.tag.bcp47 is the BCP 47 tag of the region or territory. Defined only if the locale loaded actually contains it (eg, es-MX does, but es doesn't), which is how locales behave in the CLDR. New 3.75

variant.tag.bcp47 is the BCP 47 tag of the variant (in the BCP 47 sense, like 1901 for German). New 3.75

extension.⟨s⟩.tag.bcp47 is the BCP 47 value of the extension whose singleton is ⟨s⟩ (currently the recognized singletons are x, t and u). The internal syntax can be somewhat complex, and this feature is still somewhat tentative. An example is classiclatin which sets extension.x.tag.bcp47 to classic. New 3.75

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

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

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

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

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

the macro \hechap will contain the string פרק.

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

\localeid Each language in the babel sense has its own unique numeric identifier, which can be retrieved with \localeid.

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

\LocaleForEach {⟨code⟩}

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

ensureinfo=off New 3.75 Previously, ini files are loaded only with \babelprovide and also when languages are selected if there is a \babelfont or they have not been explicitly declared. Now the ini files are loaded (and therefore the corresponding data) even if these two conditions are not met (in previous versions you had to enable it with \BabelEnsureInfo

in the preamble). Because of the way this feature works, problems are very unlikely, but there is switch as a package option to turn the new behavior off (`ensureinfo=off`).

1.20 Hyphenation and line breaking

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

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

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

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

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

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

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

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

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

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

New 3.9a Sets hyphenation exceptions for the languages given or, without the optional argument, for *all* languages (eg, proper nouns or common loan words, and of course monolingual documents). Multiple declarations work much like `\hyphenation` (last wins), but language exceptions take precedence over global ones.
It can be used only in the preamble, and exceptions are set when the language is first selected, thus taking into account changes of `\lccodes`'s done in `\extras`⟨*lang*⟩ as well as

the language-specific encoding (not set in the preamble by default). Multiple
\babelhyphenation's are allowed. For example:

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

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

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

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

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

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

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

New 3.9m *In luatex only,*[14] adds or replaces patterns for the languages given or, without
the optional argument, for *all* languages. If a pattern for a certain combination already
exists, it gets replaced by the new one.
It can be used only in the preamble, and patterns are added when the language is first
selected, thus taking into account changes of \lccodes's done in \extras⟨*lang*⟩ as well as
the language-specific encoding (not set in the preamble by default). Multiple
\babelpatterns's are allowed.
Listed patterns are saved expanded and therefore it relies on the LICR. Of course, it also
works without the LICR if the input and the font encodings are the same, like in Unicode
based engines.
New 3.31 (Only luatex.) With \babelprovide and imported CJK languages, a simple
generic line breaking algorithm (push-out-first) is applied, based on a selection of the
Unicode rules (New 3.32 it is disabled in verbatim mode, or more precisely when the
hyphenrules are set to nohyphenation). It can be activated alternatively by setting
explicitly the intraspace.
New 3.27 Interword spacing for Thai, Lao and Khemer is activated automatically if a
language with one of those scripts are loaded with \babelprovide. See the sample on the
babel repository. With both Unicode engines, spacing is based on the "current" em unit (the
size of the previous char in luatex, and the font size set by the last \selectfont in xetex).

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.

---

[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.

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

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

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

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

This applies `transliteration.omega` always, but `sigma.final` only when \withsigmafinal is set.

Here are the transforms currently predefined. (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, Oriya, Tamil, Telugu.
Latin	`digraphs.ligatures`	Replaces the groups *ae*, *AE*, *oe*, *OE* with *æ*, *Æ*, *œ*, *Œ*.
Latin	`letters.noj`	Replaces *j*, *J* with *i*, *I*.
Latin	`letters.uv`	Replaces *v*, *U* with *u*, *V*.
Sanskrit	`transliteration.iast`	The IAST system to romanize Devanagari.[16]
Serbian	`transliteration.gajica`	(Note `serbian` with `ini` files refers to the Cyrillic script, which is here the target.) The standard system devised by Ljudevit Gaj.
Arabic, Persian	`kashida.plain`	Experimental. A very simple and basic transform for 'plain' Arabic fonts, which attempts to distribute the tatwil as evenly as possible (starting at the end of the line). See the news for version 3.59.

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

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

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

In the replacements, a captured char may be mapped to another, too. For example, if the first capture reads ([ïü]), the replacement could be {1|ïü|íú}, which maps *ï* to *í*, and *ü* to *ú*, so that the diaeresis is removed.
This feature is activated with the first `\babelposthyphenation` or `\babelprehyphenation`.
New 3.67 With the optional argument you can associate a user defined transform to an attribute, so that it's active only when it's set (currently its attribute value is ignored). With this mechanism transforms can be set or unset even in the middle of paragraphs, and applied to single words. To define, set and unset the attribute, the LaTeX kernel provides the macros `\newattribute`, `\setattribute` and `\unsetattribute`. The following example shows how to use it, provided an attribute named `\latinnoj` has been declared:

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

See the babel site for a more detailed description and some examples. It also describes a few additional replacement types (`string`, `penalty`).
Although the main purpose of this command is non-standard hyphenation, it may actually be used for other transformations (after hyphenation is applied, so you must take discretionaries into account).

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

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

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

See the description above for the optional argument.

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

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

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

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

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

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

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

1.22   Selection based on BCP 47 tags

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

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

Here is a minimal example:

```
\documentclass{article}

\usepackage[danish]{babel}

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

\begin{document}

Chapter in Danish: \chaptername.

\selectlanguage{de-AT}

\localedate{2020}{1}{30}

\end{document}
```

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

`autoload.bcp47` with values `on` and `off`.

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

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

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

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

(You can deactivate it with `off`.) So, if `dutch` is one of the package (or class) options, you can write `\selectlanguage{nl}`. Note the language name does not change (in this example is still `dutch`), but you can get it with `\localeinfo` or `\getlanguageproperty`. It must be turned on explicitly for similar reasons to those explained above.

1.23   Selecting scripts

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

---

[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.

\ensureascii {⟨*text*⟩}

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

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

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

1.24 Selecting directions

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

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

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

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

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

There are some package options controlling bidi writing.

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

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

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

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

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

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

```
\documentclass{article}

\usepackage[bidi=basic]{babel}

\babelprovide[import, main]{arabic}

\babelfont{rm}{FreeSerif}

\begin{document}

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

\end{document}
```

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

```
\documentclass{book}

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

\babelprovide[onchar=ids fonts]{arabic}

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

\begin{document}

Most Arabic speakers consider the two varieties to be two registers
of one language, although the two registers can be referred to in
Arabic as فصحى العصر \textit{fuṣḥā l-'aṣr} (MSA) and
فصحى التراث \textit{fuṣḥā t-turāth} (CA).

\end{document}
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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}
```

---

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

<dl>

\babelsublr {⟨*lr-text*⟩}

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

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

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

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

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

\BabelPatchSection {⟨*section-name*⟩}

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

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

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

New 3.17 Something like:

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

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

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

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

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

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

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

</dl>

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

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

1.25 Language attributes

\languageattribute

This is a user-level command, to be used in the preamble of a document (after \usepackage[...]{babel}), that declares which attributes are to be used for a given language. It takes two arguments: the first is the name of the language; the second, a (list of) attribute(s) to be used. Attributes must be set in the preamble and only once – they cannot be turned on and off. The command checks whether the language is known in this document and whether the attribute(s) are known for this language.

Very often, using a *modifier* in a package option is better.

Several language definition files use their own methods to set options. For example, french uses \frenchsetup, magyar (1.5) uses \magyarOptions; modifiers provided by spanish have no attribute counterparts. Macros setting options are also used (eg, \ProsodicMarksOn in latin).

1.26 Hooks

New 3.9a A hook is a piece of code to be executed at certain events. Some hooks are predefined when luatex and xetex are used.

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

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

The same name can be applied to several events. Hooks with a certain {⟨name⟩} may be enabled and disabled for all defined events with \EnableBabelHook{⟨name⟩}, \DisableBabelHook{⟨name⟩}. Names containing the string babel are reserved (they are used, for example, by \useshortands* to add a hook for the event afterextras).

New 3.33 They may be also applied to a specific language with the optional argument; language-specific settings are executed after global ones.

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

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

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

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

defaultcommands Used (locally) in \StartBabelCommands.

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

stopcommands Used to reset the above, if necessary.

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

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

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

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

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

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

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

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

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

everylanguage (language) Executed before every language patterns are loaded.
loadkernel (file) By default just defines a few basic commands. It can be used to define
    different versions of them or to load a file.
loadpatterns (patterns file) Loads the patterns file. Used by luababel.def.
loadexceptions (exceptions file) Loads the exceptions file. Used by luababel.def.

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

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

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

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

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

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

1.27 Languages supported by babel with **ldf** files

In the following table most of the languages supported by babel with and .ldf file are
listed, together with the names of the option which you can load babel with for each
language. Note this list is open and the current options may be different. It does not
include ini files.

Afrikaans afrikaans
Azerbaijani azerbaijani
Basque basque
Breton breton
Bulgarian bulgarian

Catalan catalan
Croatian croatian
Czech czech
Danish danish
Dutch dutch
English english, USenglish, american, UKenglish, british, canadian, australian, newzealand
Esperanto esperanto
Estonian estonian
Finnish finnish
French french, francais, canadien, acadian
Galician galician
German austrian, german, germanb, ngerman, naustrian
Greek greek, polutonikogreek
Hebrew hebrew
Icelandic icelandic
Indonesian indonesian (bahasa, indon, bahasai)
Interlingua interlingua
Irish Gaelic irish
Italian italian
Latin latin
Lower Sorbian lowersorbian
Malay malay, melayu (bahasam)
North Sami samin
Norwegian norsk, nynorsk
Polish polish
Portuguese portuguese, brazilian (portuges, brazil)[20]
Romanian romanian
Russian russian
Scottish Gaelic scottish
Spanish spanish
Slovakian slovak
Slovenian slovene
Swedish swedish
Serbian serbian
Turkish turkish
Ukrainian ukrainian
Upper Sorbian uppersorbian
Welsh welsh

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

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.

---

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

1.28   Unicode character properties in luatex

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

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

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

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

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

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

1.29   Tweaking some features

\babeladjust   {⟨*key-value-list*⟩}

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

1.30   Tips, workarounds, known issues and notes

- If you use the document class book *and* you use \ref inside the argument of \chapter (or just use \ref inside \MakeUppercase), LaTeX will keep complaining about an undefined label. To prevent such problems, you can revert to using uppercase labels, you can use \lowercase{\ref{foo}} inside the argument of \chapter, or, if you will not use shorthands in labels, set the `safe` option to `none` or `bib`.

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

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

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

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

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

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

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

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

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

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

csquotes Logical markup for quotes.
iflang Tests correctly the current language.
hyphsubst Selects a different set of patterns for a language.
translator An open platform for packages that need to be localized.
siunitx Typesetting of numbers and physical quantities.
biblatex Programmable bibliographies and citations.
bicaption Bilingual captions.
babelbib Multilingual bibliographies.
microtype Adjusts the typesetting according to some languages (kerning and spacing). Ligatures can be disabled.
substitutefont Combines fonts in several encodings.
mkpattern Generates hyphenation patterns.
tracklang Tracks which languages have been requested.
ucharclasses (xetex) Switches fonts when you switch from one Unicode block to another.
zhspacing Spacing for CJK documents in xetex.

1.31 Current and future work

The current work is focused on the so-called complex scripts in luatex. In 8-bit engines, babel provided a basic support for bidi text as part of the style for Hebrew, but it is somewhat unsatisfactory and internally replaces some hardwired commands by other hardwired commands (generic changes would be much better).
Useful additions would be, for example, time, currency, addresses and personal names.[22]. But that is the easy part, because they don't require modifying the LaTeX internals.
Calendars (Arabic, Persian, Indic, etc.) are under study.
Also interesting are differences in the sentence structure or related to it. For example, in Basque the number precedes the name (including chapters), in Hungarian "from (1)" is "(1)-ből", but "from (3)" is "(3)-ból", in Spanish an item labelled "3.º" may be referred to as either "ítem 3.º" or "3.ᵉʳ ítem", and so on.

---

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

[22]See for example POSIX, ISO 14652 and the Unicode Common Locale Data Repository (CLDR). Those systems, however, have limited application to TeX because their aim is just to display information and not fine typesetting.

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

1.32 Tentative and experimental code

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

Options for locales loaded on the fly

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

Labels

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

2 Loading languages with `language.dat`

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

New 3.9q With luatex, however, patterns are loaded on the fly when requested by the language (except the "0th" language, typically english, which is preloaded always).[23] Until 3.9n, this task was delegated to the package luatex-hyphen, by Khaled Hosny, Élie Roux, and Manuel Pégourié-Gonnard, and required an extra file named language.dat.lua, but now a new mechanism has been devised based solely on language.dat. **You must rebuild the formats** if upgrading from a previous version. You may want to have a local language.dat for a particular project (for example, a book on Chemistry).[24]

2.1 Format

In that file the person who maintains a TeX environment has to record for which languages he has hyphenation patterns *and* in which files these are stored[25]. When hyphenation exceptions are stored in a separate file this can be indicated by naming that file *after* the file with the hyphenation patterns.

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

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

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

---

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

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

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

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

```
german:T1 hyphenT1.ger
german hyphen.ger
```

With the previous settings, if the encoding when the language is selected is T1 then the patterns in hyphenT1.ger are used, but otherwise use those in hyphen.ger (note the encoding can be set in \extras⟨*lang*⟩).
A typical error when using babel is the following:

```
No hyphenation patterns were preloaded for
the language `<lang>' into the format.
Please, configure your TeX system to add them and
rebuild the format. Now I will use the patterns
preloaded for english instead}}
```

It simply means you must reconfigure language.dat, either by hand or with the tools provided by your distribution.

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

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

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

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

- 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 /).

---

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

Some recommendations:

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

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

- Avoid adding things to \noextras⟨*lang*⟩ except for umlauthigh and friends, \bbl@deactivate, \bbl@(non)frenchspacing, and language-specific macros. Use always, if possible, \bbl@save and \bbl@savevariable (except if you still want to have access to the previous value). Do not reset a macro or a setting to a hardcoded value. Never. Instead save its value in \extras⟨*lang*⟩.

- Do not switch scripts. If you want to make sure a set of glyphs is used, switch either the font encoding (low-level) or the language (high-level, which in turn may switch the font encoding). Usage of things like \latintext is deprecated.[27]

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

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

3.1 Guidelines for contributed languages

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

---

[27]But not removed, for backward compatibility.

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

3.5 Support for saving macro definitions

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

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

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

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

3.6 Support for extending macros

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

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

3.7 Macros common to a number of languages

\bbl@allowhyphens
In several languages compound words are used. This means that when TeX has to

---

[28]This mechanism was introduced by Bernd Raichle.

hyphenate such a compound word, it only does so at the '-' that is used in such words. To allow hyphenation in the rest of such a compound word, the macro \bbl@allowhyphens can be used.

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

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

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

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

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

3.8 Encoding-dependent strings

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

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

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

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

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

The ⟨language-list⟩ specifies which languages the block is intended for. A block is taken into account only if the \CurrentOption is listed here. Alternatively, you can define \BabelLanguages to a comma-separated list of languages to be defined (if undefined, \StartBabelCommands sets it to \CurrentOption). You may write \CurrentOption as the language, but this is discouraged – a explicit name (or names) is much better and clearer. A "selector" is a name to be used as value in package option strings, optionally followed by extra info about the encodings to be used. The name unicode must be used for xetex and luatex (the key strings has also other two special values: generic and encoded). If a string is set several times (because several blocks are read), the first one takes precedence (ie, it works much like \providecommand).

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

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

Blocks without a selector are read always if the key strings has been used. They provide fallback values, and therefore must be the last blocks; they should be provided always if possible and all strings should be defined somehow inside it; they can be the only blocks

(mainly LGC scripts using the LICR). Blocks without a selector can be activated explicitly with `strings=generic` (no block is taken into account except those). With `strings=encoded`, strings in those blocks are set as default (internally, ?). With `strings=encoded` strings are protected, but they are correctly expanded in `\MakeUppercase` and the like. If there is no key `strings`, string definitions are ignored, but `\SetCases` are still honored (in a encoded way).

The ⟨*category*⟩ is either `captions`, `date` or `extras`. You must stick to these three categories, even if no error is raised when using other name.[29] It may be empty, too, but in such a case using `\SetString` is an error (but not `\SetCase`).

```
\StartBabelCommands{language}{captions}
  [unicode, fontenc=TU EU1 EU2, charset=utf8]
\SetString{\chaptername}{utf8-string}

\StartBabelCommands{language}{captions}
\SetString{\chaptername}{ascii-maybe-LICR-string}

\EndBabelCommands
```

A real example is:

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

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

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

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

\StartBabelCommands{german,austrian}{date}
  \SetString\monthiiname{Februar}
  \SetString\monthiiiname{M\"{a}rz}
  \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

---

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

to undefined and then strings are added). However, when used in the preamble or in a package, new settings are added to the previous ones, if the language exists (in the babel sense, ie, if \date⟨*language*⟩ exists).

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

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

\EndBabelCommands   Marks the end of the series of blocks.

\AfterBabelCommands   {⟨*code*⟩}

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

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

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

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

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

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

#1 is replaced by the roman numeral.

\SetCase   [⟨*map-list*⟩]{⟨*toupper-code*⟩}{⟨*tolower-code*⟩}

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}{}
```

---

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

```
\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=3.75.2751⟩⟩
2 ⟨⟨date=2022/05/28⟩⟩

Do not use the following macros in `ldf` **files. They may change in the future**. This applies mainly to those recently added for replacing, trimming and looping. The older ones, like \bbl@afterfi, will not change.
We define some basic macros which just make the code cleaner. \bbl@add is now used internally instead of \addto because of the unpredictable behavior of the latter. Used in babel.def and in babel.sty, which means in LaTeX is executed twice, but we need them when defining options and

`babel.def` cannot be load until options have been defined. This does not hurt, but should be fixed somehow.

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

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

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

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

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

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

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

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

```
39 \def\bbl@tempa#1{%
40   \long\def\bbl@trim##1##2{%
41     \futurelet\bbl@trim@a\bbl@trim@c##2\@nil\@nil#1\@nil\relax{##1}}%
42   \def\bbl@trim@c{%
```

---

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

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

\bbl@ifunset To check if a macro is defined, we create a new macro, which does the same as \@ifundefined.
However, in an ϵ-tex engine, it is based on \ifcsname, which is more efficient, and does not waste
memory.

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

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

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

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

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

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

```
88 \def\bbl@vforeach#1#2{%
89  \def\bbl@forcmd##1{#2}%
90  \bbl@fornext#1,\@nil,}
91 \def\bbl@fornext#1,{%
```

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

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

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

An extensison to the previous macro. It takes into account the parameters, and it is string based (ie, if you replace elax by ho, then \relax becomes \rho). No checking is done at all, because it is not a general purpose macro, and it is used by babel only when it works (an example where it does *not* work is in \bbl@TG@@date, and also fails if there are macros with spaces, because they are retokenized). It may change! (or even merged with \bbl@replace; I'm not sure ckecking the replacement is really necessary or just paranoia).

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

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

```
136 \def\bbl@ifsamestring#1#2{%
137   \begingroup
138     \protected@edef\bbl@tempb{#1}%
139     \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
140     \protected@edef\bbl@tempc{#2}%
141     \edef\bbl@tempc{\expandafter\strip@prefix\meaning\bbl@tempc}%
```

```
142     \ifx\bbl@tempb\bbl@tempc
143         \aftergroup\@firstoftwo
144     \else
145         \aftergroup\@secondoftwo
146     \fi
147   \endgroup}
148 \chardef\bbl@engine=%
149   \ifx\directlua\@undefined
150     \ifx\XeTeXinputencoding\@undefined
151         \z@
152     \else
153         \tw@
154     \fi
155   \else
156     \@ne
157   \fi
```

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

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

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

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

An alternative to \IfFormatAtLeastTF for old versions. Temporary.

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

The following adds some code to \extras... both before and after, while avoiding doing it twice. It's somewhat convoluted, to deal with #'s. Used to deal with alph, Alph and frenchspacing when there are already changes (with \babel@save).

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

Some files identify themselves with a LaTeX macro. The following code is placed before them to define (and then undefine) if not in LaTeX.

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

6.1 Multiple languages

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

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

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

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

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

Now we make sure all required files are loaded. When the command \AtBeginDocument doesn't exist we assume that we are dealing with a plain-based format. In that case the file plain.def is needed (which also defines \AtBeginDocument, and therefore it is not loaded twice). We need the first part when the format is created, and \orig@dump is used as a flag. Otherwise, we need to use the second part, so \orig@dump is not defined (plain.def undefines it).
Check if the current version of switch.def has been previously loaded (mainly, hyphen.cfg). If not, load it now. We cannot load babel.def here because we first need to declare and process the package options.

6.2 The Package File (LaTeX, babel.sty)

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

Start with some "private" debugging tool, and then define macros for errors.
```
212 \@ifpackagewith{babel}{debug}
213   {\providecommand\bbl@trace[1]{\message{^^J[ #1 ]}}%
214    \let\bbl@debug\@firstofone
215    \ifx\directlua\@undefined\else
216      \directlua{ Babel = Babel or {}
217        Babel.debug = true }%
218      \input{babel-debug.tex}%
219    \fi}
220   {\providecommand\bbl@trace[1]{}%
221    \let\bbl@debug\@gobble
222    \ifx\directlua\@undefined\else
223      \directlua{ Babel = Babel or {}
224        Babel.debug = false }%
225    \fi}
226 \def\bbl@error#1#2{%
227   \begingroup
```

```
228    \def\\{\MessageBreak}%
229    \PackageError{babel}{#1}{#2}%
230   \endgroup}
231 \def\bbl@warning#1{%
232   \begingroup
233    \def\\{\MessageBreak}%
234    \PackageWarning{babel}{#1}%
235   \endgroup}
236 \def\bbl@infowarn#1{%
237   \begingroup
238    \def\\{\MessageBreak}%
239    \GenericWarning
240      {(babel) \@spaces\@spaces\@spaces}%
241      {Package babel Info: #1}%
242   \endgroup}
243 \def\bbl@info#1{%
244   \begingroup
245    \def\\{\MessageBreak}%
246    \PackageInfo{babel}{#1}%
247   \endgroup}
```

This file also takes care of a number of compatibility issues with other packages an defines a few aditional package options. Apart from all the language options below we also have a few options that influence the behavior of language definition files.

Many of the following options don't do anything themselves, they are just defined in order to make it possible for babel and language definition files to check if one of them was specified by the user.

But first, include here the *Basic macros* defined above.

```
248 ⟨⟨Basic macros⟩⟩
249 \@ifpackagewith{babel}{silent}
250   {\let\bbl@info\@gobble
251    \let\bbl@infowarn\@gobble
252    \let\bbl@warning\@gobble}
253   {}
254 %
255 \def\AfterBabelLanguage#1{%
256    \global\expandafter\bbl@add\csname#1.ldf-h@@k\endcsname}%
```

If the format created a list of loaded languages (in \bbl@languages), get the name of the 0-th to show the actual language used. Also avaliable with base, because it just shows info.

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

6.3 base

The first 'real' option to be processed is base, which set the hyphenation patterns then resets ver@babel.sty so that LaTeXforgets about the first loading. After a subset of babel.def has been loaded (the old switch.def) and \AfterBabelLanguage defined, it exits.

Now the base option. With it we can define (and load, with luatex) hyphenation patterns, even if we are not interesed in the rest of babel.

66

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

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

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

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

The next option tells babel to leave shorthand characters active at the end of processing the package.
This is *not* the default as it can cause problems with other packages, but for those who want to use
the shorthand characters in the preamble of their documents this can help.

```
320 \DeclareOption{KeepShorthandsActive}{}
321 \DeclareOption{activeacute}{}
322 \DeclareOption{activegrave}{}
323 \DeclareOption{debug}{}
324 \DeclareOption{noconfigs}{}
325 \DeclareOption{showlanguages}{}
326 \DeclareOption{silent}{}
```

```
327 % \DeclareOption{mono}{}
328 \DeclareOption{shorthands=off}{\bbl@tempa shorthands=\bbl@tempa}
329 \chardef\bbl@iniflag\z@
330 \DeclareOption{provide=*}{\chardef\bbl@iniflag\@ne}     % main -> +1
331 \DeclareOption{provide+=*}{\chardef\bbl@iniflag\tw@}    % add = 2
332 \DeclareOption{provide*=*}{\chardef\bbl@iniflag\thr@@} % add + main
333 % A separate option
334 \let\bbl@autoload@options\@empty
335 \DeclareOption{provide@=*}{\def\bbl@autoload@options{import}}
336 % Don't use. Experimental. TODO.
337 \newif\ifbbl@single
338 \DeclareOption{selectors=off}{\bbl@singletrue}
339 ⟨⟨More package options⟩⟩
```

Handling of package options is done in three passes. (I [JBL] am not very happy with the idea, anyway.) The first one processes options which has been declared above or follow the syntax <key>=<value>, the second one loads the requested languages, except the main one if set with the key main, and the third one loads the latter. First, we "flag" valid keys with a nil value.

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

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

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

Now the option list is processed, taking into account only currently declared options (including those declared with a =), and <key>=<value> options (the former take precedence). Unrecognized options are saved in \bbl@language@opts, because they are language options.

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

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

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

68

6.5 Conditional loading of shorthands

If there is no shorthands=<chars>, the original babel macros are left untouched, but if there is, these macros are wrapped (in babel.def) to define only those given.

A bit of optimization: if there is no shorthands=, then \bbl@ifshorthand is always true, and it is always false if shorthands is empty. Also, some code makes sense only with shorthands=....

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

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

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

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

```
399   \edef\bbl@opt@shorthands{%
400     \expandafter\bbl@sh@string\bbl@opt@shorthands\@empty}%
```

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

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

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

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

For the option safe we use a different approach – \bbl@opt@safe says which macros are redefined (B for bibs and R for refs). By default, both are set.

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

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

```
415 \bbl@trace{Defining IfBabelLayout}
416 \ifx\bbl@opt@layout\@nnil
417   \newcommand\IfBabelLayout[3]{#3}%
418 \else
```

```
419  \newcommand\IfBabelLayout[1]{%
420    \@expandtwoargs\in@{.#1.}{.\bbl@opt@layout.}%
421    \ifin@
422      \expandafter\@firstoftwo
423    \else
424      \expandafter\@secondoftwo
425    \fi}
426 \fi
427 ⟨/package⟩
428 ⟨*core⟩
```

6.6   Interlude for Plain

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

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

That is all for the moment. Now follows some common stuff, for both Plain and LaTeX. After it, we will resume the LaTeX-only stuff.

```
436 ⟨/core⟩
437 ⟨*package | core⟩
```

7   Multiple languages

This is not a separate file (switch.def) anymore.
Plain TeX version 3.0 provides the primitive \language that is used to store the current language. When used with a pre-3.0 version this function has to be implemented by allocating a counter.

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

\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.

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

\bbl@iflanguage executes code only if the language l@ exists. Otherwise raises an error.
The argument of \bbl@fixname has to be a macro name, as it may get "fixed" if casing (lc/uc) is wrong. It's an attempt to fix a long-standing bug when \foreignlanguage and the like appear in a \MakeXXXcase. However, a lowercase form is not imposed to improve backward compatibility

(perhaps you defined a language named MYLANG, but unfortunately mixed case names cannot be trapped). Note l@ is encapsulated, so that its case does not change.

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

After a name has been 'fixed', the selectors will try to load the language. If even the fixed name is not defined, will load it on the fly, either based on its name, or if activated, its BCP47 code.
We first need a couple of macros for a simple BCP 47 look up. It also makes sure, with \bbl@bcpcase, casing is the correct one, so that sr-latn-ba becomes fr-Latn-BA. Note #4 may contain some \@empty's, but they are eventually removed. \bbl@bcplookup either returns the found ini or it is \relax.

```
474 \def\bbl@bcpcase#1#2#3#4\@@#5{%
475   \ifx\@empty#3%
476     \uppercase{\def#5{#1#2}}%
477   \else
478     \uppercase{\def#5{#1}}%
479     \lowercase{\edef#5{#5#2#3#4}}%
480   \fi}
481 \def\bbl@bcplookup#1-#2-#3-#4\@@{%
482   \let\bbl@bcp\relax
483   \lowercase{\def\bbl@tempa{#1}}%
484   \ifx\@empty#2%
485     \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
486   \else\ifx\@empty#3%
487     \bbl@bcpcase#2\@empty\@empty\@@\bbl@tempb
488     \IfFileExists{babel-\bbl@tempa-\bbl@tempb.ini}%
489       {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb}}%
490       {}%
491     \ifx\bbl@bcp\relax
492       \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
493     \fi
494   \else
495     \bbl@bcpcase#2\@empty\@empty\@@\bbl@tempb
496     \bbl@bcpcase#3\@empty\@empty\@@\bbl@tempc
497     \IfFileExists{babel-\bbl@tempa-\bbl@tempb-\bbl@tempc.ini}%
498       {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb-\bbl@tempc}}%
499       {}%
500     \ifx\bbl@bcp\relax
501       \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
502         {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
503         {}%
504     \fi
505     \ifx\bbl@bcp\relax
506       \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
507         {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
508         {}%
509     \fi
510     \ifx\bbl@bcp\relax
```

71

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

\iflanguage   Users might want to test (in a private package for instance) which language is currently active. For this we provide a test macro, \iflanguage, that has three arguments. It checks whether the first argument is a known language. If so, it compares the first argument with the value of \language. Then, depending on the result of the comparison, it executes either the second or the third argument.

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

7.1 Selecting the language

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

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

Because the command \selectlanguage could be used in a moving argument it expands to \protect\selectlanguage␣. Therefore, we have to make sure that a macro \protect exists. If it doesn't it is \let to \relax.

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

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

```
560 \let\xstring\string
```

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

\bbl@pop@language   *But* when the language change happens *inside* a group the end of the group doesn't write anything to the auxiliary files. Therefore we need TeX's aftergroup mechanism to help us. The command \aftergroup stores the token immediately following it to be executed when the current group is closed. So we define a temporary control sequence \bbl@pop@language to be executed at the end of the group. It calls \bbl@set@language with the name of the current language as its argument.

\bbl@language@stack   The previous solution works for one level of nesting groups, but as soon as more levels are used it is no longer adequate. For that case we need to keep track of the nested languages using a stack mechanism. This stack is called \bbl@language@stack and initially empty.

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

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

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

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

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

\bbl@pop@lang   This macro stores its first element (which is delimited by the '+'-sign) in \languagename and stores the rest of the string in \bbl@language@stack.

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

The reason for the somewhat weird arrangement of arguments to the helper function is the fact it is called in the following way. This means that before \bbl@pop@lang is executed TeX first *expands* the stack, stored in \bbl@language@stack. The result of that is that the argument string of \bbl@pop@lang contains one or more language names, each followed by a '+'-sign (zero language names won't occur as this macro will only be called after something has been pushed on the stack).

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

Once the name of the previous language is retrieved from the stack, it is fed to \bbl@set@language to do the actual work of switching everything that needs switching.

An alternative way to identify languages (in the babel sense) with a numerical value is introduced in 3.30. This is one of the first steps for a new interface based on the concept of locale, which explains the name of \localeid. This means \l@... will be reserved for hyphenation patterns (so that two locales can share the same rules).

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

The unprotected part of \selectlanguage.

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

\bbl@set@language   The macro \bbl@set@language takes care of switching the language environment *and* of writing
entries on the auxiliary files. For historial reasons, language names can be either language of
\language. To catch either form a trick is used, but unfortunately as a side effect the catcodes of
letters in \languagename are messed up. This is a bug, but preserved for backwards compatibility.
The list of auxiliary files can be extended by redefining \BabelContentsFiles, but make sure they
are loaded inside a group (as aux, toc, lof, and lot do) or the last language of the document will
remain active afterwards.
We also write a command to change the current language in the auxiliary files.
\bbl@savelastskip is used to deal with skips before the write whatsit (as suggested by U Fischer).
Adapted from hyperref, but it might fail, so I'll consider it a temporary hack, while I study other
options (the ideal, but very likely unfeasible except perhaps in luatex, is to avoid the \write
altogether when not needed).

```
606 \def\BabelContentsFiles{toc,lof,lot}
607 \def\bbl@set@language#1{% from selectlanguage, pop@
608   % The old buggy way. Preserved for compatibility.
609   \edef\languagename{%
610     \ifnum\escapechar=\expandafter`\string#1\@empty
611     \else\string#1\@empty\fi}%
612   \ifcat\relax\noexpand#1%
613     \expandafter\ifx\csname date\languagename\endcsname\relax
614       \edef\languagename{#1}%
615       \let\localename\languagename
616     \else
617       \bbl@info{Using '\string\language' instead of 'language' is\\%
618               deprecated. If what you want is to use a\\%
619               macro containing the actual locale, make\\%
620               sure it does not not match any language.\\%
621               Reported}%
622       \ifx\scantokens\@undefined
623         \def\localename{??}%
624       \else
625         \scantokens\expandafter{\expandafter
626           \def\expandafter\localename\expandafter{\languagename}}%
627       \fi
628     \fi
629   \else
630     \def\localename{#1}% This one has the correct catcodes
```

```
631    \fi
632    \select@language{\languagename}%
633    % write to auxs
634    \expandafter\ifx\csname date\languagename\endcsname\relax\else
635      \if@filesw
636        \ifx\babel@aux\@gobbletwo\else % Set if single in the first, redundant
637          \bbl@savelastskip
638          \protected@write\@auxout{}{\string\babel@aux{\bbl@auxname}{}}%
639          \bbl@restorelastskip
640        \fi
641        \bbl@usehooks{write}{}%
642      \fi
643    \fi}
644 %
645 \let\bbl@restorelastskip\relax
646 \let\bbl@savelastskip\relax
647 %
648 \newif\ifbbl@bcpallowed
649 \bbl@bcpallowedfalse
650 \def\select@language#1{% from set@, babel@aux
651    \ifx\bbl@selectorname\@empty
652      \def\bbl@selectorname{select}%
653    % set hymap
654    \fi
655    \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
656    % set name
657    \edef\languagename{#1}%
658    \bbl@fixname\languagename
659    % TODO. name@map must be here?
660    \bbl@provide@locale
661    \bbl@iflanguage\languagename{%
662      \expandafter\ifx\csname date\languagename\endcsname\relax
663        \bbl@error
664          {Unknown language '\languagename'. Either you have\\%
665            misspelled its name, it has not been installed,\\%
666            or you requested it in a previous run. Fix its name,\\%
667            install it or just rerun the file, respectively. In\\%
668            some cases, you may need to remove the aux file}%
669          {You may proceed, but expect wrong results}%
670      \else
671        % set type
672        \let\bbl@select@type\z@
673        \expandafter\bbl@switch\expandafter{\languagename}%
674      \fi}}
675 \def\babel@aux#1#2{%
676    \select@language{#1}%
677    \bbl@foreach\BabelContentsFiles{% \relax -> don't assume vertical mode
678      \@writefile{##1}{\babel@toc{#1}{#2}\relax}}}% TODO - plain?
679 \def\babel@toc#1#2{%
680    \select@language{#1}}
```

First, check if the user asks for a known language. If so, update the value of \language and call \originalTeX to bring TeX in a certain pre-defined state.

The name of the language is stored in the control sequence \languagename.

Then we have to *re*define \originalTeX to compensate for the things that have been activated. To save memory space for the macro definition of \originalTeX, we construct the control sequence name for the \noextras⟨lang⟩ command at definition time by expanding the \csname primitive.

Now activate the language-specific definitions. This is done by constructing the names of three macros by concatenating three words with the argument of \selectlanguage, and calling these macros.

The switching of the values of \lefthyphenmin and \righthyphenmin is somewhat different. First we save their current values, then we check if \⟨lang⟩hyphenmins is defined. If it is not, we set default values (2 and 3), otherwise the values in \⟨lang⟩hyphenmins will be used.

```
681 \newif\ifbbl@usedategroup
682 \def\bbl@switch#1{%  from select@, foreign@
683   % make sure there is info for the language if so requested
684   \bbl@ensureinfo{#1}%
685   % restore
686   \originalTeX
687   \expandafter\def\expandafter\originalTeX\expandafter{%
688     \csname noextras#1\endcsname
689     \let\originalTeX\@empty
690     \babel@beginsave}%
691   \bbl@usehooks{afterreset}{}%
692   \languageshorthands{none}%
693   % set the locale id
694   \bbl@id@assign
695   % switch captions, date
696   % No text is supposed to be added here, so we remove any
697   % spurious spaces.
698   \bbl@bsphack
699     \ifcase\bbl@select@type
700       \csname captions#1\endcsname\relax
701       \csname date#1\endcsname\relax
702     \else
703       \bbl@xin@{,captions,}{,\bbl@select@opts,}%
704       \ifin@
705         \csname captions#1\endcsname\relax
706       \fi
707       \bbl@xin@{,date,}{,\bbl@select@opts,}%
708       \ifin@  % if \foreign... within \<lang>date
709         \csname date#1\endcsname\relax
710       \fi
711     \fi
712   \bbl@esphack
713   % switch extras
714   \bbl@usehooks{beforeextras}{}%
715   \csname extras#1\endcsname\relax
716   \bbl@usehooks{afterextras}{}%
717   %  > babel-ensure
718   %  > babel-sh-<short>
719   %  > babel-bidi
720   %  > babel-fontspec
721   % hyphenation - case mapping
722   \ifcase\bbl@opt@hyphenmap\or
723     \def\BabelLower##1##2{\lccode##1=##2\relax}%
724     \ifnum\bbl@hymapsel>4\else
725       \csname\languagename @bbl@hyphenmap\endcsname
726     \fi
727     \chardef\bbl@opt@hyphenmap\z@
728   \else
729     \ifnum\bbl@hymapsel>\bbl@opt@hyphenmap\else
730       \csname\languagename @bbl@hyphenmap\endcsname
731     \fi
732   \fi
733   \let\bbl@hymapsel\@cclv
734   % hyphenation - select rules
735   \ifnum\csname l@\languagename\endcsname=\l@unhyphenated
736     \edef\bbl@tempa{u}%
737   \else
738     \edef\bbl@tempa{\bbl@cl{lnbrk}}%
739   \fi
740   % linebreaking - handle u, e, k (v in the future)
741   \bbl@xin@{/u}{/\bbl@tempa}%
742   \ifin@\else\bbl@xin@{/e}{/\bbl@tempa}\fi % elongated forms
743   \ifin@\else\bbl@xin@{/k}{/\bbl@tempa}\fi % only kashida
```

76

```
744    \ifin@\else\bbl@xin@{/v}{/\bbl@tempa}\fi % variable font
745    \ifin@
746      % unhyphenated/kashida/elongated = allow stretching
747      \language\l@unhyphenated
748      \babel@savevariable\emergencystretch
749      \emergencystretch\maxdimen
750      \babel@savevariable\hbadness
751      \hbadness\@M
752    \else
753      % other = select patterns
754      \bbl@patterns{#1}%
755    \fi
756    % hyphenation - mins
757    \babel@savevariable\lefthyphenmin
758    \babel@savevariable\righthyphenmin
759    \expandafter\ifx\csname #1hyphenmins\endcsname\relax
760      \set@hyphenmins\tw@\thr@@\relax
761    \else
762      \expandafter\expandafter\expandafter\set@hyphenmins
763        \csname #1hyphenmins\endcsname\relax
764    \fi
765    \let\bbl@selectorname\@empty}
```

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

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

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

```
771 \long\def\endotherlanguage{%
772    \global\@ignoretrue\ignorespaces}
```

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

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

At the end of the environment we need to switch off the extra definitions. The grouping mechanism of the environment will take care of resetting the correct hyphenation rules and "extras".

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

\foreignlanguage   The `\foreignlanguage` command is another substitute for the `\selectlanguage` command. This command takes two arguments, the first argument is the name of the language to use for typesetting the text specified in the second argument.
Unlike `\selectlanguage` this command doesn't switch *everything*, it only switches the hyphenation rules and the extra definitions for the language specified. It does this within a group and assumes the `\extras⟨lang⟩` command doesn't make any `\global` changes. The coding is very similar to part of `\selectlanguage`.
`\bbl@beforeforeign` is a trick to fix a bug in bidi texts. `\foreignlanguage` is supposed to be a 'text' command, and therefore it must emit a `\leavevmode`, but it does not, and therefore the indent is

placed on the opposite margin. For backward compatibility, however, it is done only if a right-to-left script is requested; otherwise, it is no-op.

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

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

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

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

\foreign@language   This macro does the work for \foreignlanguage and the otherlanguage* environment. First we need to store the name of the language and check that it is a known language. Then it just calls bbl@switch.

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

78

```
828          Reported}%
829      \fi
830      % set type
831      \let\bbl@select@type\@ne
832      \expandafter\bbl@switch\expandafter{\languagename}}}
```

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

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

\bbl@patterns   This macro selects the hyphenation patterns by changing the \language register. If special
hyphenation patterns are available specifically for the current font encoding, use them instead of the
default.

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

```
840 \let\bbl@hyphlist\@empty
841 \let\bbl@hyphenation@\relax
842 \let\bbl@pttnlist\@empty
843 \let\bbl@patterns@\relax
844 \let\bbl@hymapsel=\@cclv
845 \def\bbl@patterns#1{%
846      \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
847          \csname l@#1\endcsname
848          \edef\bbl@tempa{#1}%
849        \else
850          \csname l@#1:\f@encoding\endcsname
851          \edef\bbl@tempa{#1:\f@encoding}%
852        \fi
853      \@expandtwoargs\bbl@usehooks{patterns}{{#1}{\bbl@tempa}}%
854      %  > luatex
855      \@ifundefined{bbl@hyphenation@}{}{% Can be \relax!
856        \begingroup
857          \bbl@xin@{,\number\language,}{,\bbl@hyphlist}%
858          \ifin@\else
859            \@expandtwoargs\bbl@usehooks{hyphenation}{{#1}{\bbl@tempa}}%
860            \hyphenation{%
861              \bbl@hyphenation@
862              \@ifundefined{bbl@hyphenation@#1}%
863                \@empty
864                {\space\csname bbl@hyphenation@#1\endcsname}}%
865            \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
866          \fi
867        \endgroup}}
```

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

```
868 \def\hyphenrules#1{%
869      \edef\bbl@tempf{#1}%
870      \bbl@fixname\bbl@tempf
871      \bbl@iflanguage\bbl@tempf{%
872        \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
873        \ifx\languageshorthands\@undefined\else
874          \languageshorthands{none}%
```

```
875     \fi
876     \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax
877       \set@hyphenmins\tw@\thr@@\relax
878     \else
879       \expandafter\expandafter\expandafter\set@hyphenmins
880       \csname\bbl@tempf hyphenmins\endcsname\relax
881     \fi}}
882 \let\endhyphenrules\@empty
```

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

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

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

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

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

```
890 \ifx\ProvidesFile\@undefined
891   \def\ProvidesLanguage#1[#2 #3 #4]{%
892     \wlog{Language: #1 #4 #3 <#2>}%
893     }
894 \else
895   \def\ProvidesLanguage#1{%
896     \begingroup
897       \catcode`\ 10 %
898       \@makeother\/%
899       \@ifnextchar[%
900         {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}}
901   \def\@provideslanguage#1[#2]{%
902     \wlog{Language: #1 #2}%
903     \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
904     \endgroup}
905 \fi
```

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

```
906 \ifx\originalTeX\@undefined\let\originalTeX\@empty\fi
```

Because this part of the code can be included in a format, we make sure that the macro which
initializes the save mechanism, \babel@beginsave, is not considered to be undefined.

```
907 \ifx\babel@beginsave\@undefined\let\babel@beginsave\relax\fi
```

A few macro names are reserved for future releases of babel, which will use the concept of 'locale':

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

7.2 Errors

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

When the package was loaded without options not everything will work as expected. An error message is issued in that case.

When the format knows about \PackageError it must be LaTeX 2$_\varepsilon$, so we can safely use its error handling interface. Otherwise we'll have to 'keep it simple'.

Infos are not written to the console, but on the other hand many people think warnings are errors, so a further message type is defined: an important info which is sent to the console.

```
918 \edef\bbl@nulllanguage{\string\language=0}
919 \def\bbl@nocaption{\protect\bbl@nocaption@i}
920 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
921   \global\@namedef{#2}{\textbf{?#1?}}%
922   \@nameuse{#2}%
923   \edef\bbl@tempa{#1}%
924   \bbl@sreplace\bbl@tempa{name}{}%
925   \bbl@warning{% TODO.
926     \@backslashchar#1 not set for '\languagename'. Please,\\%
927     define it after the language has been loaded\\%
928     (typically in the preamble) with:\\%
929     \string\setlocalecaption{\languagename}{\bbl@tempa}{..}\\%
930     Reported}}
931 \def\bbl@tentative{\protect\bbl@tentative@i}
932 \def\bbl@tentative@i#1{%
933   \bbl@warning{%
934     Some functions for '#1' are tentative.\\%
935     They might not work as expected and their behavior\\%
936     could change in the future.\\%
937     Reported}}
938 \def\@nolanerr#1{%
939   \bbl@error
940     {You haven't defined the language '#1' yet.\\%
941      Perhaps you misspelled it or your installation\\%
942      is not complete}%
943    {Your command will be ignored, type <return> to proceed}}
944 \def\@nopatterns#1{%
945   \bbl@warning
946     {No hyphenation patterns were preloaded for\\%
947      the language '#1' into the format.\\%
948     Please, configure your TeX system to add them and\\%
949     rebuild the format. Now I will use the patterns\\%
950     preloaded for \bbl@nulllanguage\space instead}}
951 \let\bbl@usehooks\@gobbletwo
952 \ifx\bbl@onlyswitch\@empty\endinput\fi
953   % Here ended switch.def
```

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

```
954 \ifx\directlua\@undefined\else
955   \ifx\bbl@luapatterns\@undefined
956     \input luababel.def
957   \fi
958 \fi
959 ⟨⟨Basic macros⟩⟩
960 \bbl@trace{Compatibility with language.def}
961 \ifx\bbl@languages\@undefined
962   \ifx\directlua\@undefined
963     \openin1 = language.def % TODO. Remove hardcoded number
964     \ifeof1
965       \closein1
966       \message{I couldn't find the file language.def}
```

81

```
967        \else
968          \closein1
969          \begingroup
970            \def\addlanguage#1#2#3#4#5{%
971              \expandafter\ifx\csname lang@#1\endcsname\relax\else
972                \global\expandafter\let\csname l@#1\expandafter\endcsname
973                  \csname lang@#1\endcsname
974              \fi}%
975            \def\uselanguage#1{}%
976            \input language.def
977          \endgroup
978        \fi
979      \fi
980      \chardef\l@english\z@
981 \fi
```

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

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

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

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

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

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

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

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

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

```
1007 \def\bbl@redefinerobust#1{%
```

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

7.3 Hooks

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

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

To ensure forward compatibility, arguments in hooks are set implicitly. So, if a further argument is added in the future, there is no need to change the existing code. Note events intended for hyphen.cfg are also loaded (just in case you need them for some reason).

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

\babelensure The user command just parses the optional argument and creates a new macro named \bbl@e@⟨language⟩. We register a hook at the afterextras event which just executes this macro in a "complete" selection (which, if undefined, is \relax and does nothing). This part is somewhat involved because we have to make sure things are expanded the correct number of times. The macro \bbl@e@⟨language⟩ contains \bbl@ensure{⟨include⟩}{⟨exclude⟩}{⟨fontenc⟩}, which in in turn loops over the macros names in \bbl@captionslist, excluding (with the help of \in@) those in the exclude list. If the fontenc is given (and not \relax), the \fontencoding is also added. Then we loop over the include list, but if the macro already contains \foreignlanguage, nothing is done. Note this macro (1) is not restricted to the preamble, and (2) changes are local.

```
1047 \bbl@trace{Defining babelensure}
1048 \newcommand\babelensure[2][]{%  TODO - revise test files
1049   \AddBabelHook{babel-ensure}{afterextras}{%
1050     \ifcase\bbl@select@type
```

```
1051        \bbl@cl{e}%
1052      \fi}%
1053    \begingroup
1054      \let\bbl@ens@include\@empty
1055      \let\bbl@ens@exclude\@empty
1056      \def\bbl@ens@fontenc{\relax}%
1057      \def\bbl@tempb##1{%
1058        \ifx\@empty##1\else\noexpand##1\expandafter\bbl@tempb\fi}%
1059      \edef\bbl@tempa{\bbl@tempb#1\@empty}%
1060      \def\bbl@tempb##1=##2\@@{\@namedef{bbl@ens@##1}{##2}}%
1061      \bbl@foreach\bbl@tempa{\bbl@tempb##1\@@}%
1062      \def\bbl@tempc{\bbl@ensure}%
1063      \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1064        \expandafter{\bbl@ens@include}}%
1065      \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1066        \expandafter{\bbl@ens@exclude}}%
1067      \toks@\expandafter{\bbl@tempc}%
1068      \bbl@exp{%
1069    \endgroup
1070    \def\<bbl@e@#2>{\the\toks@{\bbl@ens@fontenc}}}}
1071 \def\bbl@ensure#1#2#3{% 1: include 2: exclude 3: fontenc
1072    \def\bbl@tempb##1{% elt for (excluding) \bbl@captionslist list
1073      \ifx##1\@undefined % 3.32 - Don't assume the macro exists
1074        \edef##1{\noexpand\bbl@nocaption
1075          {\bbl@stripslash##1}{\languagename\bbl@stripslash##1}}%
1076      \fi
1077      \ifx##1\@empty\else
1078        \in@{##1}{#2}%
1079        \ifin@\else
1080          \bbl@ifunset{bbl@ensure@\languagename}%
1081            {\bbl@exp{%
1082              \\\DeclareRobustCommand\<bbl@ensure@\languagename>[1]{%
1083                \\\foreignlanguage{\languagename}%
1084                {\ifx\relax#3\else
1085                  \\\fontencoding{#3}\\\selectfont
1086                \fi
1087                ########1}}}}%
1088            {}%
1089          \toks@\expandafter{##1}%
1090          \edef##1{%
1091            \bbl@csarg\noexpand{ensure@\languagename}%
1092            {\the\toks@}}%
1093        \fi
1094        \expandafter\bbl@tempb
1095      \fi}%
1096    \expandafter\bbl@tempb\bbl@captionslist\today\@empty
1097    \def\bbl@tempa##1{% elt for include list
1098      \ifx##1\@empty\else
1099        \bbl@csarg\in@{ensure@\languagename\expandafter}\expandafter{##1}%
1100        \ifin@\else
1101          \bbl@tempb##1\@empty
1102        \fi
1103        \expandafter\bbl@tempa
1104      \fi}%
1105    \bbl@tempa#1\@empty}
1106 \def\bbl@captionslist{%
1107    \prefacename\refname\abstractname\bibname\chaptername\appendixname
1108    \contentsname\listfigurename\listtablename\indexname\figurename
1109    \tablename\partname\enclname\ccname\headtoname\pagename\seename
1110    \alsoname\proofname\glossaryname}
```

84

7.4   Setting up language files

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

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

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

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

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

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

Finally we check \originalTeX.

```
1111 \bbl@trace{Macros for setting language files up}
1112 \def\bbl@ldfinit{%
1113   \let\bbl@screset\@empty
1114   \let\BabelStrings\bbl@opt@string
1115   \let\BabelOptions\@empty
1116   \let\BabelLanguages\relax
1117   \ifx\originalTeX\@undefined
1118     \let\originalTeX\@empty
1119   \else
1120     \originalTeX
1121   \fi}
1122 \def\LdfInit#1#2{%
1123   \chardef\atcatcode=\catcode`\@
1124   \catcode`\@=11\relax
1125   \chardef\eqcatcode=\catcode`\=
1126   \catcode`\==12\relax
1127   \expandafter\if\expandafter\@backslashchar
1128               \expandafter\@car\string#2\@nil
1129     \ifx#2\@undefined\else
1130       \ldf@quit{#1}%
1131     \fi
1132   \else
1133     \expandafter\ifx\csname#2\endcsname\relax\else
1134       \ldf@quit{#1}%
1135     \fi
1136   \fi
1137   \bbl@ldfinit}
```

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

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

\ldf@finish   This macro takes one argument. It is the name of the language that was defined in the language
definition file.

We load the local configuration file if one is present, we set the main language (taking into account
that the argument might be a control sequence that needs to be expanded) and reset the category
code of the @-sign.

```
1143 \def\bbl@afterldf#1{% TODO. Merge into the next macro? Unused elsewhere
1144   \bbl@afterlang
1145   \let\bbl@afterlang\relax
```

```
1146    \let\BabelModifiers\relax
1147    \let\bbl@screset\relax}%
1148 \def\ldf@finish#1{%
1149    \loadlocalcfg{#1}%
1150    \bbl@afterldf{#1}%
1151    \expandafter\main@language\expandafter{#1}%
1152    \catcode`\@=\atcatcode \let\atcatcode\relax
1153    \catcode`\==\eqcatcode \let\eqcatcode\relax}
```

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

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

\main@language
\bbl@main@language

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

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

We also have to make sure that some code gets executed at the beginning of the document, either when the aux file is read or, if it does not exist, when the \AtBeginDocument is executed. Languages do not set \pagedir, so we set here for the whole document to the main \bodydir.

```
1162 \def\bbl@beforestart{%
1163    \def\@nolanerr##1{%
1164      \bbl@warning{Undefined language '##1' in aux.\\Reported}}%
1165    \bbl@usehooks{beforestart}{}%
1166    \global\let\bbl@beforestart\relax}
1167 \AtBeginDocument{%
1168    {\@nameuse{bbl@beforestart}}%  Group!
1169    \if@filesw
1170      \providecommand\babel@aux[2]{}%
1171      \immediate\write\@mainaux{%
1172        \string\providecommand\string\babel@aux[2]{}}%
1173      \immediate\write\@mainaux{\string\@nameuse{bbl@beforestart}}%
1174    \fi
1175    \expandafter\selectlanguage\expandafter{\bbl@main@language}%
1176    \ifbbl@single  % must go after the line above.
1177      \renewcommand\selectlanguage[1]{}%
1178      \renewcommand\foreignlanguage[2]{#2}%
1179      \global\let\babel@aux\@gobbletwo  % Also as flag
1180    \fi
1181    \ifcase\bbl@engine\or\pagedir\bodydir\fi} % TODO - a better place
```

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

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

7.5 Shorthands

\bbl@add@special The macro \bbl@add@special is used to add a new character (or single character control sequence) to the macro \dospecials (and \@sanitize if LaTeX is used). It is used only at one place, namely when \initiate@active@char is called (which is ignored if the char has been made active before). Because \@sanitize can be undefined, we put the definition inside a conditional.
Items are added to the lists without checking its existence or the original catcode. It does not hurt, but should be fixed. It's already done with \nfss@catcodes, added in 3.10.

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

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

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

\initiate@active@char A language definition file can call this macro to make a character active. This macro takes one
argument, the character that is to be made active. When the character was already active this macro
does nothing. Otherwise, this macro defines the control sequence \normal@char⟨char⟩ to expand to
the character in its 'normal state' and it defines the active character to expand to
\normal@char⟨char⟩ by default (⟨char⟩ being the character to be made active). Later its definition
can be changed to expand to \active@char⟨char⟩ by calling \bbl@activate{⟨char⟩}.
For example, to make the double quote character active one could have \initiate@active@char{"}
in a language definition file. This defines " as \active@prefix "\active@char" (where the first " is
the character with its original catcode, when the shorthand is created, and \active@char" is a single
token). In protected contexts, it expands to \protect " or \noexpand " (ie, with the original ");
otherwise \active@char" is executed. This macro in turn expands to \normal@char" in "safe"
contexts (eg, \label), but \user@active" in normal "unsafe" ones. The latter search a definition in
the user, language and system levels, in this order, but if none is found, \normal@char" is used.
However, a deactivated shorthand (with \bbl@deactivate is defined as
\active@prefix "\normal@char".
The following macro is used to define shorthands in the three levels. It takes 4 arguments: the
(string'ed) character, \<level>@group, <level>@active and <next-level>@active (except in
system).

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

When there is also no current-level shorthand with an argument we will check whether there is a
next-level defined shorthand for this active character.

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

```
1226        \bbl@afterfi\csname#2@sh@#1@\string##1@\endcsname
1227      \fi}}%
```

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

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

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

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

If the character is already active we provide the default expansion under this shorthand mechanism. Otherwise we write a message in the transcript file, and define `\normal@char`⟨*char*⟩ to expand to the character in its default state. If the character is mathematically active when babel is loaded (for example ') the normal expansion is somewhat different to avoid an infinite loop (but it does not prevent the loop if the mathcode is set to "8000 *a posteriori*).

```
1243   \ifx#1#3\relax
1244     \expandafter\let\csname normal@char#2\endcsname#3%
1245   \else
1246     \bbl@info{Making #2 an active character}%
1247     \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
1248       \@namedef{normal@char#2}{%
1249         \textormath{#3}{\csname bbl@oridef@@#2\endcsname}}%
1250     \else
1251       \@namedef{normal@char#2}{#3}%
1252     \fi
```

To prevent problems with the loading of other packages after babel we reset the catcode of the character to the original one at the end of the package and of each language file (except with KeepShorthandsActive). It is re-activate again at \begin{document}. We also need to make sure that the shorthands are active during the processing of the .aux file. Otherwise some citations may give unexpected results in the printout when a shorthand was used in the optional argument of \bibitem for example. Then we make it active (not strictly necessary, but done for backward compatibility).

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

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

```
1262   \let\bbl@tempa\@firstoftwo
```

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

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

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

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

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

The next level of the code checks whether a user has defined a shorthand for himself with this character. First we check for a single character shorthand. If that doesn't exist we check for a shorthand with an argument.

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

In order to do the right thing when a shorthand with an argument is used by itself at the end of the line we provide a definition for the case of an empty argument. For that case we let the shorthand character expand to its non-active self. Also, When a shorthand combination such as '' ends up in a heading TeX would see \protect'\protect'. To prevent this from happening a couple of shorthand needs to be defined at user level.

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

Finally, a couple of special cases are taken care of. (1) If we are making the right quote (') active we need to change \pr@m@s as well. Also, make sure that a single ' in math mode 'does the right thing'. (2) If we are using the caret (^) as a shorthand character special care should be taken to make sure math still works. Therefore an extra level of expansion is introduced with a check for math mode on the upper level.

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

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

```
1301 ⟨⟨*More package options⟩⟩ ≡
1302 \DeclareOption{math=active}{}
```

```
1303 \DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
1304 ⟨⟨/More package options⟩⟩
```

Initiating a shorthand makes active the char. That is not strictly necessary but it is still done for backward compatibility. So we need to restore the original catcode at the end of package *and* and the end of the ldf.

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

\bbl@sh@select   This command helps the shorthand supporting macros to select how to proceed. Note that this macro needs to be expandable as do all the shorthand macros in order for them to work in expansion-only environments such as the argument of \hyphenation.
This macro expects the name of a group of shorthands in its first argument and a shorthand character in its second argument. It will expand to either \bbl@firstcs or \bbl@scndcs. Hence two more arguments need to follow it.

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

\active@prefix   The command \active@prefix which is used in the expansion of active characters has a function similar to \OT1-cmd in that it \protects the active character whenever \protect is *not* \@typeset@protect. The \@gobble is needed to remove a token such as \activechar: (when the double colon was the active character to be dealt with). There are two definitions, depending of \ifincsname is available. If there is, the expansion will be more robust.

```
1320 \begingroup
1321 \bbl@ifunset{ifincsname}% TODO. Ugly. Correct? Only Plain?
1322   {\gdef\active@prefix#1{%
1323     \ifx\protect\@typeset@protect
1324     \else
1325       \ifx\protect\@unexpandable@protect
1326         \noexpand#1%
1327       \else
1328         \protect#1%
1329       \fi
1330       \expandafter\@gobble
1331     \fi}}
1332   {\gdef\active@prefix#1{%
1333     \ifincsname
1334       \string#1%
1335       \expandafter\@gobble
1336     \else
1337       \ifx\protect\@typeset@protect
1338       \else
1339         \ifx\protect\@unexpandable@protect
1340           \noexpand#1%
1341         \else
1342           \protect#1%
1343         \fi
1344         \expandafter\expandafter\expandafter\@gobble
1345       \fi
1346     \fi}}
1347 \endgroup
```

\if@safe@actives In some circumstances it is necessary to be able to change the expansion of an active character on the fly. For this purpose the switch @safe@actives is available. The setting of this switch should be checked in the first level expansion of \active@char⟨char⟩.

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

\bbl@restore@actives When the output routine kicks in while the active characters were made "safe" this must be undone in the headers to prevent unexpected typeset results. For this situation we define a command to make them "unsafe" again.

```
1350 \def\bbl@restore@actives{\if@safe@actives\@safe@activesfalse\fi}
```

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

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

\bbl@firstcs
\bbl@scndcs These macros are used only as a trick when declaring shorthands.

```
1360 \def\bbl@firstcs#1#2{\csname#1\endcsname}
1361 \def\bbl@scndcs#1#2{\csname#2\endcsname}
```

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

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

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

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

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

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

```
1384      \else
1385        \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs
1386        \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%
1387          {\def\bbl@tempa{#4}%
1388           \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
1389           \else
1390             \bbl@info
1391               {Redefining #1 shorthand \string#2\string#3\\%
1392                in language \CurrentOption}%
1393           \fi}%
1394        \@namedef{#1@sh@\string#2@\string#3@}{#4}%
1395      \fi}
```

\textormath   Some of the shorthands that will be declared by the language definition files have to be usable in
              both text and mathmode. To achieve this the helper macro \textormath is provided.

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

\user@group   The current concept of 'shorthands' supports three levels or groups of shorthands. For each level the
\language@group name of the level or group is stored in a macro. The default is to have a user group; use language
\system@group   group 'english' and have a system group called 'system'.

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

\useshorthands   This is the user level macro. It initializes and activates the character for use as a shorthand character
                 (ie, it's active in the preamble). Languages can deactivate shorthands, so a starred version is also
                 provided which activates them always after the language has been switched.

```
1405 \def\useshorthands{%
1406   \@ifstar\bbl@usesh@s{\bbl@usesh@x{}}}
1407 \def\bbl@usesh@s#1{%
1408   \bbl@usesh@x
1409     {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bbl@activate{#1}}}%
1410     {#1}}
1411 \def\bbl@usesh@x#1#2{%
1412   \bbl@ifshorthand{#2}%
1413     {\def\user@group{user}%
1414      \initiate@active@char{#2}%
1415      #1%
1416      \bbl@activate{#2}}%
1417     {\bbl@error
1418       {I can't declare a shorthand turned off (\string#2)}
1419       {Sorry, but you can't use shorthands which have been\\%
1420        turned off in the package options}}}
```

\defineshorthand   Currently we only support two groups of user level shorthands, named internally user and
                   user@<lang> (language-dependent user shorthands). By default, only the first one is taken into
                   account, but if the former is also used (in the optional argument of \defineshorthand) a new level is
                   inserted for it (user@generic, done by \bbl@set@user@generic); we make also sure {} and
                   \protect are taken into account in this new top level.

```
1421 \def\user@language@group{user@\language@group}
1422 \def\bbl@set@user@generic#1#2{%
1423   \bbl@ifunset{user@generic@active#1}%
1424     {\bbl@active@def#1\user@language@group{user@active}{user@generic@active}%
1425      \bbl@active@def#1\user@group{user@generic@active}{language@active}%
1426      \expandafter\edef\csname#2@sh@#1@@\endcsname{%
1427        \expandafter\noexpand\csname normal@char#1\endcsname}%
1428      \expandafter\edef\csname#2@sh@#1@\string\protect@\endcsname{%
```

```
1429            \expandafter\noexpand\csname user@active#1\endcsname}}%
1430    \@empty}
1431 \newcommand\defineshorthand[3][user]{%
1432    \edef\bbl@tempa{\zap@space#1 \@empty}%
1433    \bbl@for\bbl@tempb\bbl@tempa{%
1434      \if*\expandafter\@car\bbl@tempb\@nil
1435        \edef\bbl@tempb{user@\expandafter\@gobble\bbl@tempb}%
1436        \@expandtwoargs
1437          \bbl@set@user@generic{\expandafter\string\@car#2\@nil}\bbl@tempb
1438      \fi
1439      \declare@shorthand{\bbl@tempb}{#2}{#3}}}
```

\languageshorthands A user level command to change the language from which shorthands are used. Unfortunately, babel currently does not keep track of defined groups, and therefore there is no way to catch a possible change in casing to fix it in the same way languages names are fixed. [TODO].

```
1440 \def\languageshorthands#1{\def\language@group{#1}}
```

\aliasshorthand First the new shorthand needs to be initialized. Then, we define the new shorthand in terms of the original one, but note with \aliasshorthands{"}{/} is \active@prefix /\active@char/, so we still need to let the lattest to \active@char".

```
1441 \def\aliasshorthand#1#2{%
1442    \bbl@ifshorthand{#2}%
1443      {\expandafter\ifx\csname active@char\string#2\endcsname\relax
1444        \ifx\document\@notprerr
1445          \@notshorthand{#2}%
1446        \else
1447          \initiate@active@char{#2}%
1448          \expandafter\let\csname active@char\string#2\expandafter\endcsname
1449            \csname active@char\string#1\endcsname
1450          \expandafter\let\csname normal@char\string#2\expandafter\endcsname
1451            \csname normal@char\string#1\endcsname
1452          \bbl@activate{#2}%
1453        \fi
1454      \fi}%
1455      {\bbl@error
1456        {Cannot declare a shorthand turned off (\string#2)}
1457        {Sorry, but you cannot use shorthands which have been\\%
1458         turned off in the package options}}}
```

\@notshorthand

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

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

```
1466 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
1467 \DeclareRobustCommand*\shorthandoff{%
1468    \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
1469 \def\bbl@shorthandoff#1#2{\bbl@switch@sh#1#2\@nnil}
```

\bbl@switch@sh The macro \bbl@switch@sh takes the list of characters apart one by one and subsequently switches the category code of the shorthand character according to the first argument of \bbl@switch@sh. But before any of this switching takes place we make sure that the character we are dealing with is known as a shorthand character. If it is, a macro such as \active@char" should exist.
Switching off and on is easy – we just set the category code to 'other' (12) and \active. With the starred version, the original catcode and the original definition, saved in @initiate@active@char, are restored.

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

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

```
1500 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
1501 \def\bbl@putsh#1{%
1502   \bbl@ifunset{bbl@active@\string#1}%
1503     {\bbl@putsh@i#1\@empty\@nnil}%
1504     {\csname bbl@active@\string#1\endcsname}}
1505 \def\bbl@putsh@i#1#2\@nnil{%
1506   \csname\language@group @sh@\string#1@%
1507     \ifx\@empty#2\else\string#2@\fi\endcsname}
1508 \ifx\bbl@opt@shorthands\@nnil\else
1509   \let\bbl@s@initiate@active@char\initiate@active@char
1510   \def\initiate@active@char#1{%
1511     \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
1512   \let\bbl@s@switch@sh\bbl@switch@sh
1513   \def\bbl@switch@sh#1#2{%
1514     \ifx#2\@nnil\else
1515       \bbl@afterfi
1516       \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
1517     \fi}
1518   \let\bbl@s@activate\bbl@activate
1519   \def\bbl@activate#1{%
1520     \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
1521   \let\bbl@s@deactivate\bbl@deactivate
1522   \def\bbl@deactivate#1{%
1523     \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
1524 \fi
```

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

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

\bbl@prim@s   One of the internal macros that are involved in substituting \prime for each right quote in
\bbl@pr@m@s   mathmode is \prim@s. This checks if the next character is a right quote. When the right quote is

94

active, the definition of this macro needs to be adapted to look also for an active right quote; the hat could be active, too.

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

Usually the ~ is active and expands to \penalty\@M\␣. When it is written to the .aux file it is written expanded. To prevent that and to be able to use the character ~ as a start character for a shorthand, it is redefined here as a one character shorthand on system level. The system declaration is in most cases redundant (when ~ is still a non-break space), and in some cases is inconvenient (if ~ has been redefined); however, for backward compatibility it is maintained (some existing documents may rely on the babel value).

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

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

```
1548 \expandafter\def\csname OT1dqpos\endcsname{127}
1549 \expandafter\def\csname T1dqpos\endcsname{4}
```

When the macro \f@encoding is undefined (as it is in plain TeX) we define it here to expand to OT1

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

7.6   Language attributes

Language attributes provide a means to give the user control over which features of the language definition files he wants to enable.

\languageattribute The macro \languageattribute checks whether its arguments are valid and then activates the selected language attribute. First check whether the language is known, and then process each attribute in the list.

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

We want to make sure that each attribute is selected only once; therefore we store the already selected attributes in \bbl@known@attribs. When that control sequence is not yet defined this attribute is certainly not selected before.

```
1559       \ifx\bbl@known@attribs\@undefined
```

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

When we end up here the attribute is not selected before. So, we add it to the list of selected attributes and execute the associated TₑX-code.

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

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

```
1577 \newcommand*{\@attrerr}[2]{%
1578   \bbl@error
1579     {The attribute #2 is unknown for language #1.}%
1580     {Your command will be ignored, type <return> to proceed}}
```

\bbl@declare@ttribute This command adds the new language/attribute combination to the list of known attributes. Then it defines a control sequence to be executed when the attribute is used in a document. The result of this should be that the macro \extras... for the current language is extended, otherwise the attribute will not work as its code is removed from memory at \begin{document}.

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

\bbl@ifattributeset This internal macro has 4 arguments. It can be used to interpret TₑX code based on whether a certain attribute was set. This command should appear inside the argument to \AtBeginDocument because the attributes are set in the document preamble, *after* babel is loaded.
The first argument is the language, the second argument the attribute being checked, and the third and fourth arguments are the true and false clauses.

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

\bbl@ifknown@ttrib An internal macro to check whether a given language/attribute is known. The macro takes 4 arguments, the language/attribute, the attribute list, the TₑX-code to be executed when the attribute is known and the TₑX-code to be executed otherwise.
We first assume the attribute is unknown. Then we loop over the list of known attributes, trying to find a match.

```
1599 \def\bbl@ifknown@ttrib#1#2{%
1600   \let\bbl@tempa\@secondoftwo
```

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

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

```
1608 \def\bbl@clear@ttribs{%
1609   \ifx\bbl@attributes\@undefined\else
1610     \bbl@loopx\bbl@tempa{\bbl@attributes}{%
1611       \expandafter\bbl@clear@ttrib\bbl@tempa.
1612       }%
1613     \let\bbl@attributes\@undefined
1614   \fi}
1615 \def\bbl@clear@ttrib#1-#2.{%
1616   \expandafter\let\csname#1@attr@#2\endcsname\@undefined}
1617 \AtBeginDocument{\bbl@clear@ttribs}
```

7.7 Support for saving macro definitions

To save the meaning of control sequences using \babel@save, we use temporary control sequences.
To save hash table entries for these control sequences, we don't use the name of the control sequence
to be saved to construct the temporary name. Instead we simply use the value of a counter, which is
reset to zero each time we begin to save new values. This works well because we release the saved
meanings before we begin to save a new set of control sequence meanings (see \selectlanguage
and \originalTeX). Note undefined macros are not undefined any more when saved – they are
\relax'ed.

\babel@savecnt   The initialization of a new save cycle: reset the counter to zero.
\babel@beginsave
```
1618 \bbl@trace{Macros for saving definitions}
1619 \def\babel@beginsave{\babel@savecnt\z@}
```

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

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

\babel@save   The macro \babel@save⟨csname⟩ saves the current meaning of the control sequence ⟨csname⟩ to
\babel@savevariable \originalTeX[32]. To do this, we let the current meaning to a temporary control sequence, the restore
                     commands are appended to \originalTeX and the counter is incremented. The macro
                     \babel@savevariable⟨variable⟩ saves the value of the variable. ⟨variable⟩ can be anything allowed
                     after the \the primitive.

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

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

```
1631 \def\bbl@frenchspacing{%
1632   \ifnum\the\sfcode`\.=\@m
```

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

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

7.8 Short tags

\babeltags This macro is straightforward. After zapping spaces, we loop over the list and define the macros
\text⟨*tag*⟩ and \⟨*tag*⟩. Definitions are first expanded so that they don't contain \csname but the
actual macro.

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

7.9 Hyphens

\babelhyphenation This macro saves hyphenation exceptions. Two macros are used to store them: \bbl@hyphenation@
for the global ones and \bbl@hyphenation<lang> for language ones. See \bbl@patterns above for
further details. We make sure there is a space between words when multiple commands are used.

```
1682 \bbl@trace{Hyphens}
```

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

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

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

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

```
1711 \newcommand\babelnullhyphen{\char\hyphenchar\font}
1712 \def\babelhyphen{\active@prefix\babelhyphen\bbl@hyphen}
1713 \def\bbl@hyphen{%
1714   \@ifstar{\bbl@hyphen@i @}{\bbl@hyphen@i\@empty}}
1715 \def\bbl@hyphen@i#1#2{%
1716   \bbl@ifunset{bbl@hy@#1#2\@empty}%
1717     {\csname bbl@#1usehyphen\endcsname{\discretionary{#2}{}{#2}}}%
1718     {\csname bbl@hy@#1#2\@empty\endcsname}}
```

The following two commands are used to wrap the "hyphen" and set the behavior of the rest of the word – the version with a single @ is used when further hyphenation is allowed, while that with @@ if no more hyphens are allowed. In both cases, if the hyphen is preceded by a positive space, breaking after the hyphen is disallowed.
There should not be a discretionary after a hyphen at the beginning of a word, so it is prevented if preceded by a skip. Unfortunately, this does handle cases like "(-suffix)". \nobreak is always preceded by \leavevmode, in case the shorthand starts a paragraph.

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

The following macro inserts the hyphen char.

```
1725 \def\bbl@hyphenchar{%
1726   \ifnum\hyphenchar\font=\m@ne
```

---

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

```
1727        \babelnullhyphen
1728    \else
1729        \char\hyphenchar\font
1730    \fi}
```

Finally, we define the hyphen "types". Their names will not change, so you may use them in ldf's. After a space, the \mbox in \bbl@hy@nobreak is redundant.

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

\bbl@disc   For some languages the macro \bbl@disc is used to ease the insertion of discretionaries for letters that behave 'abnormally' at a breakpoint.

```
1745 \def\bbl@disc#1#2{\nobreak\discretionary{#2-}{}{#1}\bbl@allowhyphens}
```

7.10   Multiencoding strings

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

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

```
1746 \bbl@trace{Multiencoding strings}
1747 \def\bbl@toglobal#1{\global\let#1#1}
1748 \def\bbl@recatcode#1{% TODO. Used only once?
1749    \@tempcnta="7F
1750    \def\bbl@tempa{%
1751        \ifnum\@tempcnta>"FF\else
1752            \catcode\@tempcnta=#1\relax
1753            \advance\@tempcnta\@ne
1754            \expandafter\bbl@tempa
1755        \fi}%
1756    \bbl@tempa}
```

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

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

and starts over (and similarly when lowercasing).

```
1757 \@ifpackagewith{babel}{nocase}%
1758    {\let\bbl@patchuclc\relax}%
1759    {\def\bbl@patchuclc{%
1760        \global\let\bbl@patchuclc\relax
1761        \g@addto@macro\@uclclist{\reserved@b{\reserved@b\bbl@uclc}}%
```

```
1762    \gdef\bbl@uclc##1{%
1763      \let\bbl@encoded\bbl@encoded@uclc
1764      \bbl@ifunset{\languagename @bbl@uclc}% and resumes it
1765        {##1}%
1766        {\let\bbl@tempa##1\relax % Used by LANG@bbl@uclc
1767          \csname\languagename @bbl@uclc\endcsname}%
1768      {\bbl@tolower\@empty}{\bbl@toupper\@empty}}%
1769    \gdef\bbl@tolower{\csname\languagename @bbl@lc\endcsname}%
1770    \gdef\bbl@toupper{\csname\languagename @bbl@uc\endcsname}}}
```

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

The following package options control the behavior of \SetString.

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

Main command   This is the main command. With the first use it is redefined to omit the basic
setup in subsequent blocks. We make sure strings contain actual letters in the range 128-255, not
active characters.

```
1780 \@onlypreamble\StartBabelCommands
1781 \def\StartBabelCommands{%
1782   \begingroup
1783   \bbl@recatcode{11}%
1784   ⟨⟨Macros local to BabelCommands⟩⟩
1785   \def\bbl@provstring##1##2{%
1786     \providecommand##1{##2}%
1787     \bbl@toglobal##1}%
1788   \global\let\bbl@scafter\@empty
1789   \let\StartBabelCommands\bbl@startcmds
1790   \ifx\BabelLanguages\relax
1791     \let\BabelLanguages\CurrentOption
1792   \fi
1793   \begingroup
1794   \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
1795   \StartBabelCommands}
1796 \def\bbl@startcmds{%
1797   \ifx\bbl@screset\@nnil\else
1798     \bbl@usehooks{stopcommands}{}%
1799   \fi
1800   \endgroup
1801   \begingroup
1802   \@ifstar
1803     {\ifx\bbl@opt@strings\@nnil
1804        \let\bbl@opt@strings\BabelStringsDefault
1805      \fi
1806      \bbl@startcmds@i}%
1807     \bbl@startcmds@i}
1808 \def\bbl@startcmds@i#1#2{%
1809   \edef\bbl@L{\zap@space#1 \@empty}%
1810   \edef\bbl@G{\zap@space#2 \@empty}%
1811   \bbl@startcmds@ii}
1812 \let\bbl@startcommands\StartBabelCommands
```

Parse the encoding info to get the label, input, and font parts.
Select the behavior of \SetString. Thre are two main cases, depending of if there is an optional
argument: without it and strings=encoded, strings are defined always; otherwise, they are set only
if they are still undefined (ie, fallback values). With labelled blocks and strings=encoded, define the

strings, but with another value, define strings only if the current label or font encoding is the value of `strings`; otherwise (ie, no `strings` or a block whose label is not in `strings=`) do nothing. We presume the current block is not loaded, and therefore set (above) a couple of default values to gobble the arguments. Then, these macros are redefined if necessary according to several parameters.

```
1813 \newcommand\bbl@startcmds@ii[1][\@empty]{%
1814   \let\SetString\@gobbletwo
1815   \let\bbl@stringdef\@gobbletwo
1816   \let\AfterBabelCommands\@gobble
1817   \ifx\@empty#1%
1818     \def\bbl@sc@label{generic}%
1819     \def\bbl@encstring##1##2{%
1820       \ProvideTextCommandDefault##1{##2}%
1821       \bbl@toglobal##1%
1822       \expandafter\bbl@toglobal\csname\string?\string##1\endcsname}%
1823     \let\bbl@sctest\in@true
1824   \else
1825     \let\bbl@sc@charset\space % <- zapped below
1826     \let\bbl@sc@fontenc\space % <-   "        "
1827     \def\bbl@tempa##1=##2\@nil{%
1828       \bbl@csarg\edef{sc@\zap@space##1 \@empty}{##2 }}%
1829     \bbl@vforeach{label=#1}{\bbl@tempa##1\@nil}%
1830     \def\bbl@tempa##1 ##2{% space -> comma
1831       ##1%
1832       \ifx\@empty##2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa##2\fi}%
1833     \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%
1834     \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%
1835     \edef\bbl@sc@charset{\expandafter\zap@space\bbl@sc@charset\@empty}%
1836     \def\bbl@encstring##1##2{%
1837       \bbl@foreach\bbl@sc@fontenc{%
1838         \bbl@ifunset{T@####1}%
1839           {}%
1840           {\ProvideTextCommand##1{####1}{##2}%
1841            \bbl@toglobal##1%
1842            \expandafter
1843            \bbl@toglobal\csname####1\string##1\endcsname}}}%
1844     \def\bbl@sctest{%
1845       \bbl@xin@{,\bbl@sc@label,\bbl@sc@fontenc,}}%
1846   \fi
1847   \ifx\bbl@opt@strings\@nnil        % ie, no strings key -> defaults
1848   \else\ifx\bbl@opt@strings\relax   % ie, strings=encoded
1849     \let\AfterBabelCommands\bbl@aftercmds
1850     \let\SetString\bbl@setstring
1851     \let\bbl@stringdef\bbl@encstring
1852   \else        % ie, strings=value
1853     \bbl@sctest
1854     \ifin@
1855       \let\AfterBabelCommands\bbl@aftercmds
1856       \let\SetString\bbl@setstring
1857       \let\bbl@stringdef\bbl@provstring
1858   \fi\fi\fi
1859   \bbl@scswitch
1860   \ifx\bbl@G\@empty
1861     \def\SetString##1##2{%
1862       \bbl@error{Missing group for string \string##1}%
1863         {You must assign strings to some category, typically\\%
1864          captions or extras, but you set none}}%
1865   \fi
1866   \ifx\@empty#1%
1867     \bbl@usehooks{defaultcommands}{}%
1868   \else
1869     \@expandtwoargs
1870     \bbl@usehooks{encodedcommands}{{\bbl@sc@charset}{\bbl@sc@fontenc}}%
```

```
1871    \fi}
```

There are two versions of \bbl@scswitch. The first version is used when ldfs are read, and it makes
sure \⟨group⟩⟨language⟩ is reset, but only once (\bbl@screset is used to keep track of this). The
second version is used in the preamble and packages loaded after babel and does nothing.
The macro \bbl@forlang loops \bbl@L but its body is executed only if the value is in
\BabelLanguages (inside babel) or \date⟨language⟩ is defined (after babel has been loaded). There
are also two version of \bbl@forlang. The first one skips the current iteration if the language is not
in \BabelLanguages (used in ldfs), and the second one skips undefined languages (after babel has
been loaded) .

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

Now we define commands to be used inside \StartBabelCommands.

Strings   The following macro is the actual definition of \SetString when it is "active"
First save the "switcher". Create it if undefined. Strings are defined only if undefined (ie, like
\providescommmand). With the event stringprocess you can preprocess the string by manipulating
the value of \BabelString. If there are several hooks assigned to this event, preprocessing is done in
the same order as defined. Finally, the string is set.

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

Now, some addtional stuff to be used when encoded strings are used. Captions then include
\bbl@encoded for string to be expanded in case transformations. It is \relax by default, but in
\MakeUppercase and \MakeLowercase its value is a modified expandable \@changed@cmd.

```
1909 \ifx\bbl@opt@strings\relax
1910   \def\bbl@scset#1#2{\def#1{\bbl@encoded#2}}
1911   \bbl@patchuclc
1912   \let\bbl@encoded\relax
```

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

Define \SetStringLoop, which is actually set inside \StartBabelCommands. The current definition is
somewhat complicated because we need a count, but \count@ is not under our control (remember
\SetString may call hooks). Instead of defining a dedicated count, we just "pre-expand" its value.

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

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

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

Case mapping   The command \SetCase provides a way to change the behavior of
\MakeUppercase and \MakeLowercase. \bbl@tempa is set by the patched \@uclclist to the parsing
command.

```
1941 ⟨⟨∗Macros local to BabelCommands⟩⟩ ≡
1942   \newcommand\SetCase[3][]{%
1943     \bbl@patchuclc
1944     \bbl@forlang\bbl@tempa{%
1945       \expandafter\bbl@encstring
1946         \csname\bbl@tempa @bbl@uclc\endcsname{\bbl@tempa##1}%
1947       \expandafter\bbl@encstring
1948         \csname\bbl@tempa @bbl@uc\endcsname{##2}%
1949       \expandafter\bbl@encstring
1950         \csname\bbl@tempa @bbl@lc\endcsname{##3}}}%
1951 ⟨⟨/Macros local to BabelCommands⟩⟩
```

Macros to deal with case mapping for hyphenation. To decide if the document is monolingual or
multilingual, we make a rough guess – just see if there is a comma in the languages list, built in the
first pass of the package options.

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

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

```
1958 \newcommand\BabelLower[2]{% one to one.
```

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

The following package options control the behavior of hyphenation mapping.

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

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

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

This sections ends with a general tool for resetting the caption names with a unique interface. With the old way, which mixes the switcher and the string, we convert it to the new one, which separates these two steps.

```
1995 \newcommand\setlocalecaption{%  TODO. Catch typos. What about ensure?
1996    \@ifstar\bbl@setcaption@s\bbl@setcaption@x}
1997 \def\bbl@setcaption@x#1#2#3{%  language caption-name string
1998    \bbl@trim@def\bbl@tempa{#2}%
1999    \bbl@xin@{.template}{\bbl@tempa}%
2000    \ifin@
2001      \bbl@ini@captions@template{#3}{#1}%
2002    \else
2003      \edef\bbl@tempd{%
2004        \expandafter\expandafter\expandafter
2005        \strip@prefix\expandafter\meaning\csname captions#1\endcsname}%
2006      \bbl@xin@
2007        {\expandafter\string\csname #2name\endcsname}%
2008        {\bbl@tempd}%
2009      \ifin@ % Renew caption
2010        \bbl@xin@{\string\bbl@scset}{\bbl@tempd}%
2011        \ifin@
2012          \bbl@exp{%
2013            \\\bbl@ifsamestring{\bbl@tempa}{\languagename}%
2014              {\\\bbl@scset\<#2name>\<#1#2name>}%
```

```
2015               {}}%
2016           \else % Old way converts to new way
2017             \bbl@ifunset{#1#2name}%
2018               {\bbl@exp{%
2019                 \\\bbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}%
2020                 \\\bbl@ifsamestring{\bbl@tempa}{\languagename}%
2021                   {\def\<#2name>{\<#1#2name>}}%
2022                   {}}}%
2023               {}%
2024           \fi
2025         \else
2026           \bbl@xin@{\string\bbl@scset}{\bbl@tempd}% New
2027           \ifin@ % New way
2028             \bbl@exp{%
2029               \\\bbl@add\<captions#1>{\\\bbl@scset\<#2name>\<#1#2name>}%
2030               \\\bbl@ifsamestring{\bbl@tempa}{\languagename}%
2031                 {\\\bbl@scset\<#2name>\<#1#2name>}%
2032                 {}}%
2033           \else  % Old way, but defined in the new way
2034             \bbl@exp{%
2035               \\\bbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}%
2036               \\\bbl@ifsamestring{\bbl@tempa}{\languagename}%
2037                 {\def\<#2name>{\<#1#2name>}}%
2038                 {}}%
2039           \fi%
2040         \fi
2041         \@namedef{#1#2name}{#3}%
2042         \toks@\expandafter{\bbl@captionslist}%
2043         \bbl@exp{\\\in@{\<#2name>}{\the\toks@}}%
2044         \ifin@\else
2045           \bbl@exp{\\\bbl@add\\\bbl@captionslist{\<#2name>}}%
2046           \bbl@toglobal\bbl@captionslist
2047         \fi
2048   \fi}
2049 % \def\bbl@setcaption@s#1#2#3{}  % TODO. Not yet implemented
```

7.11 Macros common to a number of languages

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

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

\save@sf@q The macro \save@sf@q is used to save and reset the current space factor.

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

7.12 Making glyphs available

This section makes a number of glyphs available that either do not exist in the OT1 encoding and have to be 'faked', or that are not accessible through T1enc.def.

7.12.1 Quotation marks

\quotedblbase In the T1 encoding the opening double quote at the baseline is available as a separate character, accessible via \quotedblbase. In the OT1 encoding it is not available, therefore we make it available by lowering the normal open quote character to the baseline.

```
2058 \ProvideTextCommand{\quotedblbase}{OT1}{%
```

```
2059    \save@sf@q{\set@low@box{\textquotedblright\/}%
2060      \box\z@\kern-.04em\bbl@allowhyphens}}
```

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

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

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

```
2063 \ProvideTextCommand{\quotesinglbase}{OT1}{%
2064    \save@sf@q{\set@low@box{\textquoteright\/}%
2065      \box\z@\kern-.04em\bbl@allowhyphens}}
```

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

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

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

```
2068 \ProvideTextCommand{\guillemetleft}{OT1}{%
2069    \ifmmode
2070      \ll
2071    \else
2072      \save@sf@q{\nobreak
2073        \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2074    \fi}
2075 \ProvideTextCommand{\guillemetright}{OT1}{%
2076    \ifmmode
2077      \gg
2078    \else
2079      \save@sf@q{\nobreak
2080        \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2081    \fi}
2082 \ProvideTextCommand{\guillemotleft}{OT1}{%
2083    \ifmmode
2084      \ll
2085    \else
2086      \save@sf@q{\nobreak
2087        \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2088    \fi}
2089 \ProvideTextCommand{\guillemotright}{OT1}{%
2090    \ifmmode
2091      \gg
2092    \else
2093      \save@sf@q{\nobreak
2094        \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2095    \fi}
```

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

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

\guilsinglleft   The single guillemets are not available in OT1 encoding. They are faked.
\guilsinglright
```
2104 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2105    \ifmmode
2106      <%
2107    \else
2108      \save@sf@q{\nobreak
```

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

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

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

7.12.2 Letters

\ij 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.

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

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

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

\dj 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).

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

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

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

For the T1 encoding \SS is defined and selects a specific glyph from the font, but for other encodings it is not available. Therefore we make it available here.

```
2155 \DeclareTextCommand{\SS}{OT1}{SS}
2156 \ProvideTextCommandDefault{\SS}{\UseTextSymbol{OT1}{\SS}}
```

7.12.3 Shorthands for quotation marks

Shorthands are provided for a number of different quotation marks, which make them usable both outside and inside mathmode. They are defined with \ProvideTextCommandDefault, but this is very likely not required because their definitions are based on encoding-dependent macros.

The 'german' single quotes.

```
2157 \ProvideTextCommandDefault{\glq}{%
2158   \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}
```

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

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

The 'german' double quotes.

```
2168 \ProvideTextCommandDefault{\glqq}{%
2169   \textormath{\quotedblbase}{\mbox{\quotedblbase}}}
```

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

```
2170 \ProvideTextCommand{\grqq}{T1}{%
2171   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2172 \ProvideTextCommand{\grqq}{TU}{%
2173   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2174 \ProvideTextCommand{\grqq}{OT1}{%
2175   \save@sf@q{\kern-.07em
2176     \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}%
2177     \kern.07em\relax}}
2178 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}
```

The 'french' single guillemets.

```
2179 \ProvideTextCommandDefault{\flq}{%
2180   \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
2181 \ProvideTextCommandDefault{\frq}{%
2182   \textormath{\guilsinglright}{\mbox{\guilsinglright}}}
```

The 'french' double guillemets.

```
2183 \ProvideTextCommandDefault{\flqq}{%
2184   \textormath{\guillemetleft}{\mbox{\guillemetleft}}}
2185 \ProvideTextCommandDefault{\frqq}{%
2186   \textormath{\guillemetright}{\mbox{\guillemetright}}}
```

7.12.4 Umlauts and tremas

The command \" needs to have a different effect for different languages. For German for instance, the 'umlaut' should be positioned lower than the default position for placing it over the letters a, o, u, A, O and U. When placed over an e, i, E or I it can retain its normal position. For Dutch the same glyph is always placed in the lower position.

\umlauthigh
\umlautlow To be able to provide both positions of \" we provide two commands to switch the positioning, the default will be \umlauthigh (the normal positioning).

```
2187 \def\umlauthigh{%
2188   \def\bbl@umlauta##1{\leavevmode\bgroup%
2189     \expandafter\accent\csname\f@encoding dqpos\endcsname
2190     ##1\bbl@allowhyphens\egroup}%
2191   \let\bbl@umlaute\bbl@umlauta}
2192 \def\umlautlow{%
2193   \def\bbl@umlauta{\protect\lower@umlaut}}
2194 \def\umlautelow{%
2195   \def\bbl@umlaute{\protect\lower@umlaut}}
2196 \umlauthigh
```

\lower@umlaut The command \lower@umlaut is used to position the \" closer to the letter.
We want the umlaut character lowered, nearer to the letter. To do this we need an extra ⟨dimen⟩ register.

```
2197 \expandafter\ifx\csname U@D\endcsname\relax
2198   \csname newdimen\endcsname\U@D
2199 \fi
```

The following code fools TEX's make_accent procedure about the current x-height of the font to force another placement of the umlaut character. First we have to save the current x-height of the font, because we'll change this font dimension and this is always done globally.
Then we compute the new x-height in such a way that the umlaut character is lowered to the base character. The value of .45ex depends on the METAFONT parameters with which the fonts were built. (Just try out, which value will look best.) If the new x-height is too low, it is not changed. Finally we call the \accent primitive, reset the old x-height and insert the base character in the argument.

```
2200 \def\lower@umlaut#1{%
2201   \leavevmode\bgroup
2202     \U@D 1ex%
2203     {\setbox\z@\hbox{%
2204       \expandafter\char\csname\f@encoding dqpos\endcsname}%
2205     \dimen@ -.45ex\advance\dimen@\ht\z@
2206     \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2207   \expandafter\accent\csname\f@encoding dqpos\endcsname
2208   \fontdimen5\font\U@D #1%
2209   \egroup}
```

For all vowels we declare \" to be a composite command which uses \bbl@umlauta or \bbl@umlaute to position the umlaut character. We need to be sure that these definitions override the ones that are provided when the package fontenc with option OT1 is used. Therefore these declarations are postponed until the beginning of the document. Note these definitions only apply to some languages, but babel sets them for *all* languages – you may want to redefine \bbl@umlauta and/or \bbl@umlaute for a language in the corresponding ldf (using the babel switching mechanism, of course).

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

Finally, make sure the default hyphenrules are defined (even if empty). For internal use, another empty \language is defined. Currently used in Amharic.

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

```
2226 \ifx\l@unhyphenated\@undefined
2227   \newlanguage\l@unhyphenated
2228 \fi
```

7.13 Layout

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

```
2229 \bbl@trace{Bidi layout}
2230 \providecommand\IfBabelLayout[3]{#3}%
2231 \newcommand\BabelPatchSection[1]{%
2232   \@ifundefined{#1}{}{%
2233     \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
2234     \@namedef{#1}{%
2235       \@ifstar{\bbl@presec@s{#1}}%
2236              {\@dblarg{\bbl@presec@x{#1}}}}}}
2237 \def\bbl@presec@x#1[#2]#3{%
2238   \bbl@exp{%
2239     \\\select@language@x{\bbl@main@language}%
2240     \\\bbl@cs{sspre@#1}%
2241     \\\bbl@cs{ss@#1}%
2242     [\\\foreignlanguage{\languagename}{\unexpanded{#2}}]%
2243     {\\\foreignlanguage{\languagename}{\unexpanded{#3}}}%
2244     \\\select@language@x{\languagename}}}
2245 \def\bbl@presec@s#1#2{%
2246   \bbl@exp{%
2247     \\\select@language@x{\bbl@main@language}%
2248     \\\bbl@cs{sspre@#1}%
2249     \\\bbl@cs{ss@#1}*%
2250     {\\\foreignlanguage{\languagename}{\unexpanded{#2}}}%
2251     \\\select@language@x{\languagename}}}
2252 \IfBabelLayout{sectioning}%
2253   {\BabelPatchSection{part}%
2254    \BabelPatchSection{chapter}%
2255    \BabelPatchSection{section}%
2256    \BabelPatchSection{subsection}%
2257    \BabelPatchSection{subsubsection}%
2258    \BabelPatchSection{paragraph}%
2259    \BabelPatchSection{subparagraph}%
2260    \def\babel@toc#1{%
2261      \select@language@x{\bbl@main@language}}}{}
2262 \IfBabelLayout{captions}%
2263   {\BabelPatchSection{caption}}{}
```

7.14 Load engine specific macros

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

7.15 Creating and modifying languages

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

```
2272 \bbl@trace{Creating languages and reading ini files}
2273 \let\bbl@extend@ini\@gobble
2274 \newcommand\babelprovide[2][]{%
2275   \let\bbl@savelangname\languagename
```

```
2276    \edef\bbl@savelocaleid{\the\localeid}%
2277    % Set name and locale id
2278    \edef\languagename{#2}%
2279    \bbl@id@assign
2280    % Initialize keys
2281    \let\bbl@KVP@captions\@nil
2282    \let\bbl@KVP@date\@nil
2283    \let\bbl@KVP@import\@nil
2284    \let\bbl@KVP@main\@nil
2285    \let\bbl@KVP@script\@nil
2286    \let\bbl@KVP@language\@nil
2287    \let\bbl@KVP@hyphenrules\@nil
2288    \let\bbl@KVP@linebreaking\@nil
2289    \let\bbl@KVP@justification\@nil
2290    \let\bbl@KVP@mapfont\@nil
2291    \let\bbl@KVP@maparabic\@nil
2292    \let\bbl@KVP@mapdigits\@nil
2293    \let\bbl@KVP@intraspace\@nil
2294    \let\bbl@KVP@intrapenalty\@nil
2295    \let\bbl@KVP@onchar\@nil
2296    \let\bbl@KVP@transforms\@nil
2297    \global\let\bbl@release@transforms\@empty
2298    \let\bbl@KVP@alph\@nil
2299    \let\bbl@KVP@Alph\@nil
2300    \let\bbl@KVP@labels\@nil
2301    \bbl@csarg\let{KVP@labels*}\@nil
2302    \global\let\bbl@inidata\@empty
2303    \global\let\bbl@extend@ini\@gobble
2304    \gdef\bbl@key@list{;}%
2305    \bbl@forkv{#1}{%  TODO - error handling
2306      \in@{/}{##1}%
2307      \ifin@
2308        \global\let\bbl@extend@ini\bbl@extend@ini@aux
2309        \bbl@renewinikey##1\@@{##2}%
2310      \else
2311        \bbl@csarg\def{KVP@##1}{##2}%
2312      \fi}%
2313    \chardef\bbl@howloaded=% 0:none; 1:ldf without ini; 2:ini
2314      \bbl@ifunset{date#2}\z@{\bbl@ifunset{bbl@llevel@#2}\@ne\tw@}%
2315    % == init ==
2316    \ifx\bbl@screset\@undefined
2317      \bbl@ldfinit
2318    \fi
2319    % ==
2320    \let\bbl@lbkflag\relax % \@empty = do setup linebreak
2321    \ifcase\bbl@howloaded
2322      \let\bbl@lbkflag\@empty % new
2323    \else
2324      \ifx\bbl@KVP@hyphenrules\@nil\else
2325        \let\bbl@lbkflag\@empty
2326      \fi
2327      \ifx\bbl@KVP@import\@nil\else
2328        \let\bbl@lbkflag\@empty
2329      \fi
2330    \fi
2331    % == import, captions ==
2332    \ifx\bbl@KVP@import\@nil\else
2333      \bbl@exp{\\\bbl@ifblank{\bbl@KVP@import}}%
2334        {\ifx\bbl@initoload\relax
2335          \begingroup
2336            \def\BabelBeforeIni##1##2{\gdef\bbl@KVP@import{##1}\endinput}%
2337            \bbl@input@texini{#2}%
2338          \endgroup
```

```
2339        \else
2340          \xdef\bbl@KVP@import{\bbl@initoload}%
2341        \fi}%
2342      {}%
2343    \fi
2344    \ifx\bbl@KVP@captions\@nil
2345      \let\bbl@KVP@captions\bbl@KVP@import
2346    \fi
2347    % ==
2348    \ifx\bbl@KVP@transforms\@nil\else
2349      \bbl@replace\bbl@KVP@transforms{ }{,}%
2350    \fi
2351    % == Load ini ==
2352    \ifcase\bbl@howloaded
2353      \bbl@provide@new{#2}%
2354    \else
2355      \bbl@ifblank{#1}%
2356        {}%  With \bbl@load@basic below
2357        {\bbl@provide@renew{#2}}%
2358    \fi
2359    % Post tasks
2360    % ----------
2361    % == subsequent calls after the first provide for a locale ==
2362    \ifx\bbl@inidata\@empty\else
2363      \bbl@extend@ini{#2}%
2364    \fi
2365    % == ensure captions ==
2366    \ifx\bbl@KVP@captions\@nil\else
2367      \bbl@ifunset{bbl@extracaps@#2}%
2368        {\bbl@exp{\\\babelensure[exclude=\\\today]{#2}}}%
2369        {\bbl@exp{\\\babelensure[exclude=\\\today,
2370                  include=\[bbl@extracaps@#2]]{#2}}}%
2371      \bbl@ifunset{bbl@ensure@\languagename}%
2372        {\bbl@exp{%
2373          \\\DeclareRobustCommand\<bbl@ensure@\languagename>[1]{%
2374            \\\foreignlanguage{\languagename}%
2375            {####1}}}}%
2376        {}%
2377      \bbl@exp{%
2378        \\\bbl@toglobal\<bbl@ensure@\languagename>%
2379        \\\bbl@toglobal\<bbl@ensure@\languagename\space>}%
2380    \fi
2381    % ==
2382    % At this point all parameters are defined if 'import'. Now we
2383    % execute some code depending on them. But what about if nothing was
2384    % imported? We just set the basic parameters, but still loading the
2385    % whole ini file.
2386    \bbl@load@basic{#2}%
2387    % == script, language ==
2388    % Override the values from ini or defines them
2389    \ifx\bbl@KVP@script\@nil\else
2390      \bbl@csarg\edef{sname@#2}{\bbl@KVP@script}%
2391    \fi
2392    \ifx\bbl@KVP@language\@nil\else
2393      \bbl@csarg\edef{lname@#2}{\bbl@KVP@language}%
2394    \fi
2395    \ifcase\bbl@engine\or
2396      \bbl@ifunset{bbl@chrng@\languagename}{}%
2397        {\directlua{
2398          Babel.set_chranges_b('\bbl@cl{sbcp}', '\bbl@cl{chrng}') }}%
2399    \fi
2400     % == onchar ==
2401    \ifx\bbl@KVP@onchar\@nil\else
```

```
2402    \bbl@luahyphenate
2403    \bbl@exp{%
2404      \\\AddToHook{env/document/before}{{\\\select@language{#2}{}}}}%
2405    \directlua{
2406      if Babel.locale_mapped == nil then
2407        Babel.locale_mapped = true
2408        Babel.linebreaking.add_before(Babel.locale_map)
2409        Babel.loc_to_scr = {}
2410        Babel.chr_to_loc = Babel.chr_to_loc or {}
2411      end}%
2412    \bbl@xin@{ ids }{ \bbl@KVP@onchar\space}%
2413    \ifin@
2414      \ifx\bbl@starthyphens\@undefined % Needed if no explicit selection
2415        \AddBabelHook{babel-onchar}{beforestart}{{\bbl@starthyphens}}%
2416      \fi
2417      \bbl@exp{\\\bbl@add\\\bbl@starthyphens
2418        {\\\bbl@patterns@lua{\languagename}}}%
2419      % TODO - error/warning if no script
2420      \directlua{
2421        if Babel.script_blocks['\bbl@cl{sbcp}'] then
2422          Babel.loc_to_scr[\the\localeid] =
2423            Babel.script_blocks['\bbl@cl{sbcp}']
2424          Babel.locale_props[\the\localeid].lc = \the\localeid\space
2425          Babel.locale_props[\the\localeid].lg = \the\@nameuse{l@\languagename}\space
2426        end
2427      }%
2428    \fi
2429    \bbl@xin@{ fonts }{ \bbl@KVP@onchar\space}%
2430    \ifin@
2431      \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
2432      \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
2433      \directlua{
2434        if Babel.script_blocks['\bbl@cl{sbcp}'] then
2435          Babel.loc_to_scr[\the\localeid] =
2436            Babel.script_blocks['\bbl@cl{sbcp}']
2437        end}%
2438      \ifx\bbl@mapselect\@undefined  % TODO. almost the same as mapfont
2439        \AtBeginDocument{%
2440          \bbl@patchfont{{\bbl@mapselect}}%
2441          {\selectfont}}%
2442        \def\bbl@mapselect{%
2443          \let\bbl@mapselect\relax
2444          \edef\bbl@prefontid{\fontid\font}}%
2445        \def\bbl@mapdir##1{%
2446          {\def\languagename{##1}%
2447           \let\bbl@ifrestoring\@firstoftwo % To avoid font warning
2448           \bbl@switchfont
2449           \ifnum\fontid\font>\z@ % A hack, for the pgf nullfont hack
2450             \directlua{
2451               Babel.locale_props[\the\csname bbl@id@@##1\endcsname]%
2452                       ['/\bbl@prefontid'] = \fontid\font\space}%
2453          \fi}}%
2454      \fi
2455      \bbl@exp{\\\bbl@add\\\bbl@mapselect{\\\bbl@mapdir{\languagename}}}%
2456    \fi
2457    % TODO - catch non-valid values
2458    \fi
2459    % == mapfont ==
2460    % For bidi texts, to switch the font based on direction
2461    \ifx\bbl@KVP@mapfont\@nil\else
2462      \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}{}%
2463        {\bbl@error{Option '\bbl@KVP@mapfont' unknown for\\%
2464                   mapfont. Use 'direction'.%
```

```
2465                {See the manual for details.}}}%
2466      \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
2467      \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
2468      \ifx\bbl@mapselect\@undefined % TODO. See onchar.
2469        \AtBeginDocument{%
2470          \bbl@patchfont{{\bbl@mapselect}}%
2471          {\selectfont}}%
2472        \def\bbl@mapselect{%
2473          \let\bbl@mapselect\relax
2474          \edef\bbl@prefontid{\fontid\font}}%
2475        \def\bbl@mapdir##1{%
2476          {\def\languagename{##1}%
2477           \let\bbl@ifrestoring\@firstoftwo % avoid font warning
2478           \bbl@switchfont
2479           \directlua{Babel.fontmap
2480             [\the\csname bbl@wdir@##1\endcsname]%
2481             [\bbl@prefontid]=\fontid\font}}}%
2482      \fi
2483      \bbl@exp{\\\bbl@add\\\bbl@mapselect{\\\bbl@mapdir{\languagename}}}%
2484    \fi
2485    % == Line breaking: intraspace, intrapenalty ==
2486    % For CJK, East Asian, Southeast Asian, if interspace in ini
2487    \ifx\bbl@KVP@intraspace\@nil\else % We can override the ini or set
2488      \bbl@csarg\edef{intsp@#2}{\bbl@KVP@intraspace}%
2489    \fi
2490    \bbl@provide@intraspace
2491    % == Line breaking: CJK quotes ==
2492    \ifcase\bbl@engine\or
2493      \bbl@xin@{/c}{/\bbl@cl{lnbrk}}%
2494      \ifin@
2495        \bbl@ifunset{bbl@quote@\languagename}{}%
2496          {\directlua{
2497            Babel.locale_props[\the\localeid].cjk_quotes = {}
2498            local cs = 'op'
2499            for c in string.utfvalues(%
2500              [[\csname bbl@quote@\languagename\endcsname]]) do
2501            if Babel.cjk_characters[c].c == 'qu' then
2502              Babel.locale_props[\the\localeid].cjk_quotes[c] = cs
2503            end
2504            cs = ( cs == 'op') and 'cl' or 'op'
2505          end
2506        }}%
2507      \fi
2508    \fi
2509    % == Line breaking: justification ==
2510    \ifx\bbl@KVP@justification\@nil\else
2511      \let\bbl@KVP@linebreaking\bbl@KVP@justification
2512    \fi
2513    \ifx\bbl@KVP@linebreaking\@nil\else
2514      \bbl@xin@{,\bbl@KVP@linebreaking,}{,elongated,kashida,cjk,unhyphenated,}%
2515      \ifin@
2516        \bbl@csarg\xdef
2517          {lnbrk@\languagename}{\expandafter\@car\bbl@KVP@linebreaking\@nil}%
2518      \fi
2519    \fi
2520    \bbl@xin@{/e}{/\bbl@cl{lnbrk}}%
2521    \ifin@\else\bbl@xin@{/k}{/\bbl@cl{lnbrk}}\fi
2522    \ifin@\bbl@arabicjust\fi
2523    % == Line breaking: hyphenate.other.(locale|script) ==
2524    \ifx\bbl@lbkflag\@empty
2525      \bbl@ifunset{bbl@hyotl@\languagename}{}%
2526        {\bbl@csarg\bbl@replace{hyotl@\languagename}{ }{,}%
2527         \bbl@startcommands*{\languagename}{}%
```

```
2528         \bbl@csarg\bbl@foreach{hyotl@\languagename}{%
2529           \ifcase\bbl@engine
2530             \ifnum##1<257
2531               \SetHyphenMap{\BabelLower{##1}{##1}}%
2532             \fi
2533           \else
2534             \SetHyphenMap{\BabelLower{##1}{##1}}%
2535           \fi}%
2536         \bbl@endcommands}%
2537       \bbl@ifunset{bbl@hyots@\languagename}{}%
2538         {\bbl@csarg\bbl@replace{hyots@\languagename}{ }{,}%
2539          \bbl@csarg\bbl@foreach{hyots@\languagename}{%
2540            \ifcase\bbl@engine
2541              \ifnum##1<257
2542                \global\lccode##1=##1\relax
2543              \fi
2544            \else
2545              \global\lccode##1=##1\relax
2546            \fi}}%
2547   \fi
2548   % == Counters: maparabic ==
2549   % Native digits, if provided in ini (TeX level, xe and lua)
2550   \ifcase\bbl@engine\else
2551     \bbl@ifunset{bbl@dgnat@\languagename}{}%
2552       {\expandafter\ifx\csname bbl@dgnat@\languagename\endcsname\@empty\else
2553         \expandafter\expandafter\expandafter
2554         \bbl@setdigits\csname bbl@dgnat@\languagename\endcsname
2555         \ifx\bbl@KVP@maparabic\@nil\else
2556           \ifx\bbl@latinarabic\@undefined
2557             \expandafter\let\expandafter\@arabic
2558               \csname bbl@counter@\languagename\endcsname
2559           \else    % ie, if layout=counters, which redefines \@arabic
2560             \expandafter\let\expandafter\bbl@latinarabic
2561               \csname bbl@counter@\languagename\endcsname
2562           \fi
2563         \fi
2564       \fi}%
2565   \fi
2566   % == Counters: mapdigits ==
2567   % Native digits (lua level).
2568   \ifodd\bbl@engine
2569     \ifx\bbl@KVP@mapdigits\@nil\else
2570       \bbl@ifunset{bbl@dgnat@\languagename}{}%
2571         {\RequirePackage{luatexbase}%
2572          \bbl@activate@preotf
2573          \directlua{
2574            Babel = Babel or {}  %%% -> presets in luababel
2575            Babel.digits_mapped = true
2576            Babel.digits = Babel.digits or {}
2577            Babel.digits[\the\localeid] =
2578              table.pack(string.utfvalue('\bbl@cl{dgnat}'))
2579            if not Babel.numbers then
2580              function Babel.numbers(head)
2581                local LOCALE = Babel.attr_locale
2582                local GLYPH = node.id'glyph'
2583                local inmath = false
2584                for item in node.traverse(head) do
2585                  if not inmath and item.id == GLYPH then
2586                    local temp = node.get_attribute(item, LOCALE)
2587                    if Babel.digits[temp] then
2588                      local chr = item.char
2589                      if chr > 47 and chr < 58 then
2590                        item.char = Babel.digits[temp][chr-47]
```

```
2591                    end
2592                  end
2593                elseif item.id == node.id'math' then
2594                  inmath = (item.subtype == 0)
2595                end
2596              end
2597            return head
2598          end
2599        end
2600      }}%
2601    \fi
2602  \fi
2603  % == Counters: alph, Alph ==
2604  % What if extras<lang> contains a \babel@save\@alph? It won't be
2605  % restored correctly when exiting the language, so we ignore
2606  % this change with the \bbl@alph@saved trick.
2607  \ifx\bbl@KVP@alph\@nil\else
2608    \bbl@extras@wrap{\\\bbl@alph@saved}%
2609      {\let\bbl@alph@saved\@alph}%
2610      {\let\@alph\bbl@alph@saved
2611      \babel@save\@alph}%
2612    \bbl@exp{%
2613      \\\bbl@add\<extras\languagename>{%
2614        \let\\\@alph\<bbl@cntr@\bbl@KVP@alph @\languagename>}}%
2615  \fi
2616  \ifx\bbl@KVP@Alph\@nil\else
2617    \bbl@extras@wrap{\\\bbl@Alph@saved}%
2618      {\let\bbl@Alph@saved\@Alph}%
2619      {\let\@Alph\bbl@Alph@saved
2620      \babel@save\@Alph}%
2621    \bbl@exp{%
2622      \\\bbl@add\<extras\languagename>{%
2623        \let\\\@Alph\<bbl@cntr@\bbl@KVP@Alph @\languagename>}}%
2624  \fi
2625  % == require.babel in ini ==
2626  % To load or reaload the babel-*.tex, if require.babel in ini
2627  \ifx\bbl@beforestart\relax\else  % But not in doc aux or body
2628    \bbl@ifunset{bbl@rqtex@\languagename}{}%
2629      {\expandafter\ifx\csname bbl@rqtex@\languagename\endcsname\@empty\else
2630        \let\BabelBeforeIni\@gobbletwo
2631        \chardef\atcatcode=\catcode`\@
2632        \catcode`\@=11\relax
2633        \bbl@input@texini{\bbl@cs{rqtex@\languagename}}%
2634        \catcode`\@=\atcatcode
2635        \let\atcatcode\relax
2636        \global\bbl@csarg\let{rqtex@\languagename}\relax
2637      \fi}%
2638    \bbl@ifunset{bbl@rqcal@\languagename}{}%
2639      {\edef\bbl@tempa{\bbl@cs{rqcal@\languagename}}%
2640        \bbl@replace\bbl@tempa{ }{,}%
2641        \bbl@foreach\bbl@tempa{%
2642          \bbl@ifunset{bbl@ca@##1}{%
2643            \chardef\atcatcode=\catcode`\@
2644            \catcode`\@=11\relax
2645            \InputIfFileExists{babel-ca-##1.tex}{}{}%
2646            \catcode`\@=\atcatcode
2647            \let\atcatcode\relax}%
2648        {}}}%
2649  \fi
2650  % == frenchspacing ==
2651  \ifcase\bbl@howloaded\in@true\else\in@false\fi
2652  \ifin@\else\bbl@xin@{typography/frenchspacing}{\bbl@key@list}\fi
2653  \ifin@
```

```
2654        \bbl@extras@wrap{\\\bbl@pre@fs}%
2655          {\bbl@pre@fs}%
2656          {\bbl@post@fs}%
2657      \fi
2658      % == Release saved transforms ==
2659      \bbl@release@transforms\relax % \relax closes the last item.
2660      % == main ==
2661      \ifx\bbl@KVP@main\@nil  % Restore only if not 'main'
2662        \let\languagename\bbl@savelangname
2663        \chardef\localeid\bbl@savelocaleid\relax
2664      \fi}
```

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

```
2665 \def\bbl@provide@new#1{%
2666   \@namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
2667   \@namedef{extras#1}{}%
2668   \@namedef{noextras#1}{}%
2669   \bbl@startcommands*{#1}{captions}%
2670     \ifx\bbl@KVP@captions\@nil %        and also if import, implicit
2671       \def\bbl@tempb##1{%              elt for \bbl@captionslist
2672         \ifx##1\@empty\else
2673           \bbl@exp{%
2674             \\\SetString\\##1{%
2675               \\\bbl@nocaption{\bbl@stripslash##1}{#1\bbl@stripslash##1}}}%
2676           \expandafter\bbl@tempb
2677         \fi}%
2678       \expandafter\bbl@tempb\bbl@captionslist\@empty
2679     \else
2680       \ifx\bbl@initoload\relax
2681         \bbl@read@ini{\bbl@KVP@captions}2%  % Here letters cat = 11
2682       \else
2683         \bbl@read@ini{\bbl@initoload}2%      % Same
2684       \fi
2685     \fi
2686   \StartBabelCommands*{#1}{date}%
2687     \ifx\bbl@KVP@import\@nil
2688       \bbl@exp{%
2689         \\\SetString\\\today{\\\bbl@nocaption{today}{#1today}}}%
2690     \else
2691       \bbl@savetoday
2692       \bbl@savedate
2693     \fi
2694   \bbl@endcommands
2695   \bbl@load@basic{#1}%
2696   % == hyphenmins == (only if new)
2697   \bbl@exp{%
2698     \gdef\<#1hyphenmins>{%
2699       {\bbl@ifunset{bbl@lfthm@#1}{2}{\bbl@cs{lfthm@#1}}}%
2700       {\bbl@ifunset{bbl@rgthm@#1}{3}{\bbl@cs{rgthm@#1}}}}}%
2701   % == hyphenrules (also in renew) ==
2702   \bbl@provide@hyphens{#1}%
2703   \ifx\bbl@KVP@main\@nil\else
2704     \expandafter\main@language\expandafter{#1}%
2705   \fi}
2706 %
2707 \def\bbl@provide@renew#1{%
2708   \ifx\bbl@KVP@captions\@nil\else
2709     \StartBabelCommands*{#1}{captions}%
2710       \bbl@read@ini{\bbl@KVP@captions}2%    % Here all letters cat = 11
2711     \EndBabelCommands
2712   \fi
2713   \ifx\bbl@KVP@import\@nil\else
```

```
2714     \StartBabelCommands*{#1}{date}%
2715       \bbl@savetoday
2716       \bbl@savedate
2717     \EndBabelCommands
2718   \fi
2719   % == hyphenrules (also in new) ==
2720   \ifx\bbl@lbkflag\@empty
2721     \bbl@provide@hyphens{#1}%
2722   \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.)

```
2723 \def\bbl@load@basic#1{%
2724   \ifcase\bbl@howloaded\or\or
2725     \ifcase\csname bbl@llevel@\languagename\endcsname
2726       \bbl@csarg\let{lname@\languagename}\relax
2727     \fi
2728   \fi
2729   \bbl@ifunset{bbl@lname@#1}%
2730     {\def\BabelBeforeIni##1##2{%
2731        \begingroup
2732          \let\bbl@ini@captions@aux\@gobbletwo
2733          \def\bbl@inidate ####1.####2.####3.####4\relax ####5####6{}%
2734          \bbl@read@ini{##1}1%
2735          \ifx\bbl@initoload\relax\endinput\fi
2736        \endgroup}%
2737      \begingroup        % boxed, to avoid extra spaces:
2738        \ifx\bbl@initoload\relax
2739          \bbl@input@texini{#1}%
2740        \else
2741          \setbox\z@\hbox{\BabelBeforeIni{\bbl@initoload}{}}%
2742        \fi
2743      \endgroup}%
2744     {}}
```

The hyphenrules option is handled with an auxiliary macro.

```
2745 \def\bbl@provide@hyphens#1{%
2746   \let\bbl@tempa\relax
2747   \ifx\bbl@KVP@hyphenrules\@nil\else
2748     \bbl@replace\bbl@KVP@hyphenrules{ }{,}%
2749     \bbl@foreach\bbl@KVP@hyphenrules{%
2750       \ifx\bbl@tempa\relax     % if not yet found
2751         \bbl@ifsamestring{##1}{+}%
2752           {{\bbl@exp{\\\addlanguage\<l@##1>}}}%
2753           {}%
2754         \bbl@ifunset{l@##1}%
2755           {}%
2756           {\bbl@exp{\let\bbl@tempa\<l@##1>}}%
2757       \fi}%
2758   \fi
2759   \ifx\bbl@tempa\relax %          if no opt or no language in opt found
2760     \ifx\bbl@KVP@import\@nil
2761       \ifx\bbl@initoload\relax\else
2762         \bbl@exp{%               and hyphenrules is not empty
2763           \\\bbl@ifblank{\bbl@cs{hyphr@#1}}%
2764             {}%
2765             {\let\\\bbl@tempa\<l@\bbl@cl{hyphr}>}}%
2766       \fi
2767     \else % if importing
2768       \bbl@exp{%                  and hyphenrules is not empty
2769         \\\bbl@ifblank{\bbl@cs{hyphr@#1}}%
2770           {}%
2771           {\let\\\bbl@tempa\<l@\bbl@cl{hyphr}>}}%
```

```
2772      \fi
2773    \fi
2774    \bbl@ifunset{bbl@tempa}%          ie, relax or undefined
2775      {\bbl@ifunset{l@#1}%            no hyphenrules found - fallback
2776         {\bbl@exp{\\\adddialect\<l@#1>\language}}%
2777         {}}%                         so, l@<lang> is ok - nothing to do
2778      {\bbl@exp{\\\adddialect\<l@#1>\bbl@tempa}}}% found in opt list or ini
```

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

```
2779 \def\bbl@input@texini#1{%
2780    \bbl@bsphack
2781      \bbl@exp{%
2782         \catcode`\\\%=14 \catcode`\\\\=0
2783         \catcode`\\\{=1  \catcode`\\\}=2
2784         \lowercase{\\\InputIfFileExists{babel-#1.tex}{}{}}%
2785         \catcode`\\\%=\the\catcode`\%\relax
2786         \catcode`\\\\=\the\catcode`\\\relax
2787         \catcode`\\\{=\the\catcode`\{\relax
2788         \catcode`\\\}=\the\catcode`\}\relax}%
2789    \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.

```
2790 \def\bbl@iniline#1\bbl@iniline{%
2791    \@ifnextchar[\bbl@inisect{\@ifnextchar;\bbl@iniskip\bbl@inistore}#1\@@}% ]
2792 \def\bbl@inisect[#1]#2\@@{\def\bbl@section{#1}}
2793 \def\bbl@iniskip#1\@@{}%        if starts with ;
2794 \def\bbl@inistore#1=#2\@@{%     full (default)
2795    \bbl@trim@def\bbl@tempa{#1}%
2796    \bbl@trim\toks@{#2}%
2797    \bbl@xin@{;\bbl@section/\bbl@tempa;}{\bbl@key@list}%
2798    \ifin@\else
2799      \bbl@exp{%
2800         \\\g@addto@macro\\\bbl@inidata{%
2801            \\\bbl@elt{\bbl@section}{\bbl@tempa}{\the\toks@}}}%
2802    \fi}
2803 \def\bbl@inistore@min#1=#2\@@{%  minimal (maybe set in \bbl@read@ini)
2804    \bbl@trim@def\bbl@tempa{#1}%
2805    \bbl@trim\toks@{#2}%
2806    \bbl@xin@{.identification.}{.\bbl@section.}%
2807    \ifin@
2808      \bbl@exp{\\\g@addto@macro\\\bbl@inidata{%
2809         \\\bbl@elt{identification}{\bbl@tempa}{\the\toks@}}}%
2810    \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.

```
2811 \ifx\bbl@readstream\@undefined
2812    \csname newread\endcsname\bbl@readstream
2813 \fi
2814 \def\bbl@read@ini#1#2{%
2815    \global\let\bbl@extend@ini\@gobble
2816    \openin\bbl@readstream=babel-#1.ini
2817    \ifeof\bbl@readstream
2818      \bbl@error
2819        {There is no ini file for the requested language\\%
2820         (#1: \languagename). Perhaps you misspelled it or your\\%
2821         installation is not complete.}%
2822        {Fix the name or reinstall babel.}%
```

```
2823    \else
2824      % == Store ini data in \bbl@inidata ==
2825      \catcode`\[=12 \catcode`\]=12 \catcode`\==12 \catcode`\&=12
2826      \catcode`\;=12 \catcode`\|=12 \catcode`\%=14 \catcode`\-=12
2827      \bbl@info{Importing
2828                     \ifcase#2font and identification \or basic \fi
2829                     data for \languagename\\%
2830                 from babel-#1.ini. Reported}%
2831      \ifnum#2=\z@
2832        \global\let\bbl@inidata\@empty
2833        \let\bbl@inistore\bbl@inistore@min    % Remember it's local
2834      \fi
2835      \def\bbl@section{identification}%
2836      \bbl@exp{\\\bbl@inistore tag.ini=#1\\\@@}%
2837      \bbl@inistore load.level=#2\@@
2838      \loop
2839      \if T\ifeof\bbl@readstream F\fi T\relax % Trick, because inside \loop
2840        \endlinechar\m@ne
2841        \read\bbl@readstream to \bbl@line
2842        \endlinechar`\^^M
2843        \ifx\bbl@line\@empty\else
2844          \expandafter\bbl@iniline\bbl@line\bbl@iniline
2845        \fi
2846      \repeat
2847      % == Process stored data ==
2848      \bbl@csarg\xdef{lini@\languagename}{#1}%
2849      \bbl@read@ini@aux
2850      % == 'Export' data ==
2851      \bbl@ini@exports{#2}%
2852      \global\bbl@csarg\let{inidata@\languagename}\bbl@inidata
2853      \global\let\bbl@inidata\@empty
2854      \bbl@exp{\\\bbl@add@list\\\bbl@ini@loaded{\languagename}}%
2855      \bbl@toglobal\bbl@ini@loaded
2856    \fi}
2857 \def\bbl@read@ini@aux{%
2858    \let\bbl@savestrings\@empty
2859    \let\bbl@savetoday\@empty
2860    \let\bbl@savedate\@empty
2861    \def\bbl@elt##1##2##3{%
2862      \def\bbl@section{##1}%
2863      \in@{=date.}{=##1}% Find a better place
2864      \ifin@
2865        \bbl@ini@calendar{##1}%
2866      \fi
2867      \in@{=identification/extension.}{=##1/##2}%
2868      \ifin@
2869        \bbl@ini@extension{##2}%
2870      \fi
2871      \bbl@ifunset{bbl@inikv@##1}{}%
2872        {\csname bbl@inikv@##1\endcsname{##2}{##3}}}%
2873    \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.

```
2874 \def\bbl@extend@ini@aux#1{%
2875    \bbl@startcommands*{#1}{captions}%
2876      % Activate captions/... and modify exports
2877      \bbl@csarg\def{inikv@captions.licr}##1##2{%
2878        \setlocalecaption{#1}{##1}{##2}}%
2879      \def\bbl@inikv@captions##1##2{%
2880        \bbl@ini@captions@aux{##1}{##2}}%
2881      \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2882      \def\bbl@exportkey##1##2##3{%
```

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

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

```
2901 \def\bbl@ini@calendar#1{%
2902   \lowercase{\def\bbl@tempa{=#1=}}%
2903   \bbl@replace\bbl@tempa{=date.gregorian}{}%
2904   \bbl@replace\bbl@tempa{=date.}{}%
2905   \in@{.licr=}{#1=}%
2906   \ifin@
2907     \ifcase\bbl@engine
2908       \bbl@replace\bbl@tempa{.licr=}{}%
2909     \else
2910       \let\bbl@tempa\relax
2911     \fi
2912   \fi
2913   \ifx\bbl@tempa\relax\else
2914     \bbl@replace\bbl@tempa{=}{}%
2915     \bbl@exp{%
2916       \def\<bbl@inikv@#1>####1####2{%
2917         \\\bbl@inidate####1...\relax{####2}{\bbl@tempa}}}%
2918   \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).

```
2919 \def\bbl@renewinikey#1/#2\@@#3{%
2920   \edef\bbl@tempa{\zap@space #1 \@empty}%   section
2921   \edef\bbl@tempb{\zap@space #2 \@empty}%   key
2922   \bbl@trim\toks@{#3}%                      value
2923   \bbl@exp{%
2924     \edef\\\bbl@key@list{\bbl@key@list \bbl@tempa/\bbl@tempb;}%
2925     \\\g@addto@macro\\\bbl@inidata{%
2926       \\\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.

```
2927 \def\bbl@exportkey#1#2#3{%
2928   \bbl@ifunset{bbl@@kv@#2}%
2929     {\bbl@csarg\gdef{#1@\languagename}{#3}}%
2930     {\expandafter\ifx\csname bbl@@kv@#2\endcsname\@empty
2931       \bbl@csarg\gdef{#1@\languagename}{#3}%
2932     \else
2933       \bbl@exp{\global\let\<bbl@#1@\languagename>\<bbl@@kv@#2>}%
2934     \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.

```
2935 \def\bbl@iniwarning#1{%
2936   \bbl@ifunset{bbl@@kv@identification.warning#1}{}%
2937     {\bbl@warning{%
2938        From babel-\bbl@cs{lini@\languagename}.ini:\\%
2939        \bbl@cs{@kv@identification.warning#1}\\%
2940        Reported }}}
2941 %
2942 \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.

```
2943 \def\bbl@ini@extension#1{%
2944   \def\bbl@tempa{#1}%
2945   \bbl@replace\bbl@tempa{extension.}{}%
2946   \bbl@replace\bbl@tempa{.tag.bcp47}{}%
2947   \bbl@ifunset{bbl@info@#1}%
2948     {\bbl@csarg\xdef{info@#1}{ext/\bbl@tempa}%
2949     \bbl@exp{%
2950       \\\g@addto@macro\\\bbl@moreinfo{%
2951         \\\bbl@exportkey{ext/\bbl@tempa}{identification.#1}{}}}}%
2952     {}}
2953 \let\bbl@moreinfo\@empty
2954 %
2955 \def\bbl@ini@exports#1{%
2956   % Identification always exported
2957   \bbl@iniwarning{}%
2958   \ifcase\bbl@engine
2959     \bbl@iniwarning{.pdflatex}%
2960   \or
2961     \bbl@iniwarning{.lualatex}%
2962   \or
2963     \bbl@iniwarning{.xelatex}%
2964   \fi%
2965   \bbl@exportkey{llevel}{identification.load.level}{}%
2966   \bbl@exportkey{elname}{identification.name.english}{}%
2967   \bbl@exp{\\\bbl@exportkey{lname}{identification.name.opentype}%
2968     {\csname bbl@elname@\languagename\endcsname}}%
2969   \bbl@exportkey{tbcp}{identification.tag.bcp47}{}%
2970   \bbl@exportkey{lbcp}{identification.language.tag.bcp47}{}%
2971   \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
2972   \bbl@exportkey{esname}{identification.script.name}{}%
2973   \bbl@exp{\\\bbl@exportkey{sname}{identification.script.name.opentype}%
2974     {\csname bbl@esname@\languagename\endcsname}}%
2975   \bbl@exportkey{sbcp}{identification.script.tag.bcp47}{}%
2976   \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
2977   \bbl@exportkey{rbcp}{identification.region.tag.bcp47}{}%
2978   \bbl@exportkey{vbcp}{identification.variant.tag.bcp47}{}%
2979   \bbl@moreinfo
2980   % Also maps bcp47 -> languagename
2981   \ifbbl@bcptoname
2982     \bbl@csarg\xdef{bcp@map@\bbl@cl{tbcp}}{\languagename}%
2983   \fi
2984   % Conditional
2985   \ifnum#1>\z@         % 0 = only info, 1, 2 = basic, (re)new
2986     \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
2987     \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
2988     \bbl@exportkey{lfthm}{typography.lefthyphenmin}{2}%
2989     \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
2990     \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
2991     \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
```

```
2992    \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
2993    \bbl@exportkey{intsp}{typography.intraspace}{}%
2994    \bbl@exportkey{frspc}{typography.frenchspacing}{u}%
2995    \bbl@exportkey{chrng}{characters.ranges}{}%
2996    \bbl@exportkey{quote}{characters.delimiters.quotes}{}%
2997    \bbl@exportkey{dgnat}{numbers.digits.native}{}%
2998    \ifnum#1=\tw@            % only (re)new
2999      \bbl@exportkey{rqtex}{identification.require.babel}{}%
3000      \bbl@exportkey{rqcal}{identification.require.calendars}{}%
3001      \bbl@toglobal\bbl@savetoday
3002      \bbl@toglobal\bbl@savedate
3003      \bbl@savestrings
3004    \fi
3005  \fi}
```

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

```
3006 \def\bbl@inikv#1#2{%      key=value
3007   \toks@{#2}%              This hides #'s from ini values
3008   \bbl@csarg\edef{@kv@\bbl@section.#1}{\the\toks@}}
```

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

```
3009 \let\bbl@inikv@identification\bbl@inikv
3010 \let\bbl@inikv@typography\bbl@inikv
3011 \let\bbl@inikv@characters\bbl@inikv
3012 \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'.

```
3013 \def\bbl@inikv@counters#1#2{%
3014   \bbl@ifsamestring{#1}{digits}%
3015     {\bbl@error{The counter name 'digits' is reserved for mapping\\%
3016                 decimal digits}%
3017                {Use another name.}}%
3018     {}%
3019   \def\bbl@tempc{#1}%
3020   \bbl@trim@def{\bbl@tempb*}{#2}%
3021   \in@{.1$}{#1$}%
3022   \ifin@
3023     \bbl@replace\bbl@tempc{.1}{}%
3024     \bbl@csarg\protected@xdef{cntr@\bbl@tempc @\languagename}{%
3025       \noexpand\bbl@alphnumeral{\bbl@tempc}}%
3026   \fi
3027   \in@{.F.}{#1}%
3028   \ifin@\else\in@{.S.}{#1}\fi
3029   \ifin@
3030     \bbl@csarg\protected@xdef{cntr@#1@\languagename}{\bbl@tempb*}%
3031   \else
3032     \toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa
3033     \expandafter\bbl@buildifcase\bbl@tempb* \\ % Space after \\
3034     \bbl@csarg{\global\expandafter\let}{cntr@#1@\languagename}\bbl@tempa
3035   \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.

```
3036 \ifcase\bbl@engine
3037   \bbl@csarg\def{inikv@captions.licr}#1#2{%
3038     \bbl@ini@captions@aux{#1}{#2}}
3039 \else
3040   \def\bbl@inikv@captions#1#2{%
3041     \bbl@ini@captions@aux{#1}{#2}}
3042 \fi
```

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

```
3043 \def\bbl@ini@captions@template#1#2{% string language tempa=capt-name
3044   \bbl@replace\bbl@tempa{.template}{}%
3045   \def\bbl@toreplace{#1{}}%
3046   \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
3047   \bbl@replace\bbl@toreplace{[[}{\csname}%
3048   \bbl@replace\bbl@toreplace{[}{\csname the}%
3049   \bbl@replace\bbl@toreplace{]]}{name\endcsname{}}%
3050   \bbl@replace\bbl@toreplace{]}{\endcsname{}}%
3051   \bbl@xin@{,\bbl@tempa,}{,chapter,appendix,part,}%
3052   \ifin@
3053     \@nameuse{bbl@patch\bbl@tempa}%
3054     \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
3055   \fi
3056   \bbl@xin@{,\bbl@tempa,}{,figure,table,}%
3057   \ifin@
3058     \toks@\expandafter{\bbl@toreplace}%
3059     \bbl@exp{\gdef\<fnum@\bbl@tempa>{\the\toks@}}%
3060   \fi}
3061 \def\bbl@ini@captions@aux#1#2{%
3062   \bbl@trim@def\bbl@tempa{#1}%
3063   \bbl@xin@{.template}{\bbl@tempa}%
3064   \ifin@
3065     \bbl@ini@captions@template{#2}\languagename
3066   \else
3067     \bbl@ifblank{#2}%
3068       {\bbl@exp{%
3069         \toks@{\\\bbl@nocaption{\bbl@tempa}{\languagename\bbl@tempa name}}}}%
3070       {\bbl@trim\toks@{#2}}%
3071     \bbl@exp{%
3072       \\\bbl@add\\\bbl@savestrings{%
3073         \\\SetString\<\bbl@tempa name>{\the\toks@}}}%
3074     \toks@\expandafter{\bbl@captionslist}%
3075     \bbl@exp{\\\in@{\<\bbl@tempa name>}{\the\toks@}}%
3076     \ifin@\else
3077       \bbl@exp{%
3078         \\\bbl@add\<bbl@extracaps@\languagename>{\<\bbl@tempa name>}%
3079         \\\bbl@toglobal\<bbl@extracaps@\languagename>}%
3080     \fi
3081   \fi}
```

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

```
3082 \def\bbl@list@the{%
3083   part,chapter,section,subsection,subsubsection,paragraph,%
3084   subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,%
3085   table,page,footnote,mpfootnote,mpfn}
3086 \def\bbl@map@cnt#1{% #1:roman,etc, // #2:enumi,etc
3087   \bbl@ifunset{bbl@map@#1@\languagename}%
3088     {\@nameuse{#1}}%
3089     {\@nameuse{bbl@map@#1@\languagename}}}
3090 \def\bbl@inikv@labels#1#2{%
3091   \in@{.map}{#1}%
3092   \ifin@
3093     \ifx\bbl@KVP@labels\@nil\else
3094       \bbl@xin@{ map }{ \bbl@KVP@labels\space}%
3095       \ifin@
3096         \def\bbl@tempc{#1}%
3097         \bbl@replace\bbl@tempc{.map}{}%
3098         \in@{,#2,}{,arabic,roman,Roman,alph,Alph,fnsymbol,}%
3099         \bbl@exp{%
3100           \gdef\<bbl@map@\bbl@tempc @\languagename>%
3101             {\ifin@\<#2>\else\\\localcounter{#2}\fi}}%
3102         \bbl@foreach\bbl@list@the{%
```

```
3103        \bbl@ifunset{the##1}{}%
3104          {\bbl@exp{\let\\\bbl@tempd\<the##1>}%
3105           \bbl@exp{%
3106             \\\bbl@sreplace\<the##1>%
3107               {\<\bbl@tempc>{##1}}{\\\bbl@map@cnt{\bbl@tempc}{##1}}%
3108             \\\bbl@sreplace\<the##1>%
3109               {\<\@empty @\bbl@tempc>\<c@##1>}{\\\bbl@map@cnt{\bbl@tempc}{##1}}}%
3110           \expandafter\ifx\csname the##1\endcsname\bbl@tempd\else
3111             \toks@\expandafter\expandafter\expandafter{%
3112               \csname the##1\endcsname}%
3113             \expandafter\xdef\csname the##1\endcsname{{\the\toks@}}%
3114           \fi}}%
3115      \fi
3116    \fi
3117  %
3118  \else
3119    %
3120    % The following code is still under study. You can test it and make
3121    % suggestions. Eg, enumerate.2 = ([enumi]).([enumii]). It's
3122    % language dependent.
3123    \in@{enumerate.}{#1}%
3124    \ifin@
3125      \def\bbl@tempa{#1}%
3126      \bbl@replace\bbl@tempa{enumerate.}{}%
3127      \def\bbl@toreplace{#2}%
3128      \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
3129      \bbl@replace\bbl@toreplace{[}{\csname the}%
3130      \bbl@replace\bbl@toreplace{]}{\endcsname{}}%
3131      \toks@\expandafter{\bbl@toreplace}%
3132      % TODO. Execute only once:
3133      \bbl@exp{%
3134        \\\bbl@add\<extras\languagename>{%
3135          \\\babel@save\<labelenum\romannumeral\bbl@tempa>%
3136          \def\<labelenum\romannumeral\bbl@tempa>{\the\toks@}}%
3137        \\\bbl@toglobal\<extras\languagename>}%
3138    \fi
3139  \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.

```
3140 \def\bbl@chaptype{chapter}
3141 \ifx\@makechapterhead\@undefined
3142   \let\bbl@patchchapter\relax
3143 \else\ifx\thechapter\@undefined
3144   \let\bbl@patchchapter\relax
3145 \else\ifx\ps@headings\@undefined
3146   \let\bbl@patchchapter\relax
3147 \else
3148   \def\bbl@patchchapter{%
3149     \global\let\bbl@patchchapter\relax
3150     \gdef\bbl@chfmt{%
3151       \bbl@ifunset{bbl@\bbl@chaptype fmt@\languagename}%
3152         {\@chapapp\space\thechapter}
3153         {\@nameuse{bbl@\bbl@chaptype fmt@\languagename}}}%
3154     \bbl@add\appendix{\def\bbl@chaptype{appendix}}% Not harmful, I hope
3155     \bbl@sreplace\ps@headings{\@chapapp\ \thechapter}{\bbl@chfmt}%
3156     \bbl@sreplace\chaptermark{\@chapapp\ \thechapter}{\bbl@chfmt}%
3157     \bbl@sreplace\@makechapterhead{\@chapapp\space\thechapter}{\bbl@chfmt}%
3158     \bbl@toglobal\appendix
3159     \bbl@toglobal\ps@headings
3160     \bbl@toglobal\chaptermark
```

```
3161      \bbl@toglobal\@makechapterhead}
3162      \let\bbl@patchappendix\bbl@patchchapter
3163 \fi\fi\fi
3164 \ifx\@part\@undefined
3165   \let\bbl@patchpart\relax
3166 \else
3167   \def\bbl@patchpart{%
3168     \global\let\bbl@patchpart\relax
3169     \gdef\bbl@partformat{%
3170       \bbl@ifunset{bbl@partfmt@\languagename}%
3171         {\partname\nobreakspace\thepart}
3172         {\@nameuse{bbl@partfmt@\languagename}}}%
3173     \bbl@sreplace\@part{\partname\nobreakspace\thepart}{\bbl@partformat}%
3174     \bbl@toglobal\@part}
3175 \fi
```

Date. TODO. Document

```
3176 % Arguments are _not_ protected.
3177 \let\bbl@calendar\@empty
3178 \DeclareRobustCommand\localedate[1][]{\bbl@localedate{#1}}
3179 \def\bbl@localedate#1#2#3#4{%
3180   \begingroup
3181     \ifx\@empty#1\@empty\else
3182       \let\bbl@ld@calendar\@empty
3183       \let\bbl@ld@variant\@empty
3184       \edef\bbl@tempa{\zap@space#1 \@empty}%
3185       \def\bbl@tempb##1=##2\@@{\@namedef{bbl@ld@##1}{##2}}%
3186       \bbl@foreach\bbl@tempa{\bbl@tempb##1\@@}%
3187       \edef\bbl@calendar{%
3188         \bbl@ld@calendar
3189         \ifx\bbl@ld@variant\@empty\else
3190           .\bbl@ld@variant
3191         \fi}%
3192       \bbl@replace\bbl@calendar{gregorian}{}%
3193     \fi
3194     \bbl@cased
3195       {\@nameuse{bbl@date@\languagename @\bbl@calendar}{#2}{#3}{#4}}%
3196   \endgroup}
3197 % eg: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
3198 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
3199   \bbl@trim@def\bbl@tempa{#1.#2}%
3200   \bbl@ifsamestring{\bbl@tempa}{months.wide}%        to savedate
3201     {\bbl@trim@def\bbl@tempa{#3}%
3202     \bbl@trim\toks@{#5}%
3203     \@temptokena\expandafter{\bbl@savedate}%
3204     \bbl@exp{%   Reverse order - in ini last wins
3205       \def\\\bbl@savedate{%
3206         \\\SetString\<month\romannumeral\bbl@tempa#6name>{\the\toks@}%
3207         \the\@temptokena}}}%
3208     {\bbl@ifsamestring{\bbl@tempa}{date.long}%      defined now
3209     {\lowercase{\def\bbl@tempb{#6}}%
3210     \bbl@trim@def\bbl@toreplace{#5}%
3211     \bbl@TG@@date
3212     \bbl@ifunset{bbl@date@\languagename @}%
3213       {\bbl@exp{% TODO. Move to a better place.
3214         \gdef\<\languagename date>{\\\protect\<\languagename date >}%
3215         \gdef\<\languagename date >####1####2####3{%
3216           \\\bbl@usedategrouptrue
3217           \<bbl@ensure@\languagename>{%
3218             \\\localedate{####1}{####2}{####3}}}%
3219         \\\bbl@add\\\bbl@savetoday{%
3220           \\\SetString\\\today{%
3221             \<\languagename date>%
```

```
3222                    {\\\the\year}{\\\the\month}{\\\the\day}}}}}%
3223              {}%
3224           \global\bbl@csarg\let{date@\languagename @}\bbl@toreplace
3225           \ifx\bbl@tempb\@empty\else
3226              \global\bbl@csarg\let{date@\languagename @\bbl@tempb}\bbl@toreplace
3227           \fi}%
3228         {}}}
```

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).

```
3229 \let\bbl@calendar\@empty
3230 \newcommand\babelcalendar[2][\the\year-\the\month-\the\day]{%
3231   \@nameuse{bbl@ca@#2}#1\@@}
3232 \newcommand\BabelDateSpace{\nobreakspace}
3233 \newcommand\BabelDateDot{.\@}  % TODO. \let instead of repeating
3234 \newcommand\BabelDated[1]{{\number#1}}
3235 \newcommand\BabelDatedd[1]{{\ifnum#1<10 0\fi\number#1}}
3236 \newcommand\BabelDateM[1]{{\number#1}}
3237 \newcommand\BabelDateMM[1]{{\ifnum#1<10 0\fi\number#1}}
3238 \newcommand\BabelDateMMMM[1]{{%
3239   \csname month\romannumeral#1\bbl@calendar name\endcsname}}%
3240 \newcommand\BabelDatey[1]{{\number#1}}%
3241 \newcommand\BabelDateyy[1]{{%
3242   \ifnum#1<10 0\number#1 %
3243   \else\ifnum#1<100 \number#1 %
3244   \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3245   \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
3246   \else
3247     \bbl@error
3248       {Currently two-digit years are restricted to the\\
3249        range 0-9999.}%
3250       {There is little you can do. Sorry.}%
3251   \fi\fi\fi\fi}}
3252 \newcommand\BabelDateyyyy[1]{{\number#1}} % TODO - add leading 0
3253 \def\bbl@replace@finish@iii#1{%
3254   \bbl@exp{\def\\#1####1####2####3{\the\toks@}}}
3255 \def\bbl@TG@@date{%
3256   \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace{}}%
3257   \bbl@replace\bbl@toreplace{[.]}{\BabelDateDot{}}%
3258   \bbl@replace\bbl@toreplace{[d]}{\BabelDated{####3}}%
3259   \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{####3}}%
3260   \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{####2}}%
3261   \bbl@replace\bbl@toreplace{[MM]}{\BabelDateMM{####2}}%
3262   \bbl@replace\bbl@toreplace{[MMMM]}{\BabelDateMMMM{####2}}%
3263   \bbl@replace\bbl@toreplace{[y]}{\BabelDatey{####1}}%
3264   \bbl@replace\bbl@toreplace{[yy]}{\BabelDateyy{####1}}%
3265   \bbl@replace\bbl@toreplace{[yyyy]}{\BabelDateyyyy{####1}}%
3266   \bbl@replace\bbl@toreplace{[y|}{\bbl@datecntr[####1|}%
3267   \bbl@replace\bbl@toreplace{[m|}{\bbl@datecntr[####2|}%
3268   \bbl@replace\bbl@toreplace{[d|}{\bbl@datecntr[####3|}%
3269   \bbl@replace@finish@iii\bbl@toreplace}
3270 \def\bbl@datecntr{\expandafter\bbl@xdatecntr\expandafter}
3271 \def\bbl@xdatecntr[#1|#2]{\localenumeral{#2}{#1}}
```

Transforms.

```
3272 \let\bbl@release@transforms\@empty
3273 \@namedef{bbl@inikv@transforms.prehyphenation}{%
3274   \bbl@transforms\babelprehyphenation}
3275 \@namedef{bbl@inikv@transforms.posthyphenation}{%
3276   \bbl@transforms\babelposthyphenation}
3277 \def\bbl@transforms@aux#1#2#3#4,#5\relax{%
```

```
3278   #1[#2]{#3}{#4}{#5}}
3279 \begingroup %  A hack. TODO. Don't require an specific order
3280   \catcode`\%=12
3281   \catcode`\&=14
3282   \gdef\bbl@transforms#1#2#3{&%
3283     \ifx\bbl@KVP@transforms\@nil\else
3284       \directlua{
3285         local str = [==[#2]==]
3286         str = str:gsub('%.%d+%.%d+$', '')
3287         tex.print([[\def\string\babeltempa{]] .. str .. [[}]])
3288       }&%
3289     \bbl@xin@{,\babeltempa,}{,\bbl@KVP@transforms,}&%
3290     \ifin@
3291       \in@{.0$}{#2$}&%
3292       \ifin@
3293         \directlua{
3294           local str = string.match([[\bbl@KVP@transforms]],
3295                         '%(([^%(]-)%)[^%)]-\babeltempa')
3296           if str == nil then
3297             tex.print([[\def\string\babeltempb{}]])
3298           else
3299             tex.print([[\def\string\babeltempb{,attribute=]] .. str .. [[}]])
3300           end
3301         }
3302         \toks@{#3}&%
3303         \bbl@exp{&%
3304           \\\g@addto@macro\\\bbl@release@transforms{&%
3305             \relax  &% Closes previous \bbl@transforms@aux
3306             \\\bbl@transforms@aux
3307               \\#1{label=\babeltempa\babeltempb}{\languagename}{\the\toks@}}}&%
3308       \else
3309         \g@addto@macro\bbl@release@transforms{, {#3}}&%
3310       \fi
3311     \fi
3312   \fi}
3313 \endgroup
```

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

```
3314 \def\bbl@provide@lsys#1{%
3315   \bbl@ifunset{bbl@lname@#1}%
3316     {\bbl@load@info{#1}}%
3317     {}%
3318   \bbl@csarg\let{lsys@#1}\@empty
3319   \bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}{}%
3320   \bbl@ifunset{bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{}%
3321   \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
3322   \bbl@ifunset{bbl@lname@#1}{}%
3323     {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
3324   \ifcase\bbl@engine\or\or
3325     \bbl@ifunset{bbl@prehc@#1}{}%
3326       {\bbl@exp{\\\bbl@ifblank{\bbl@cs{prehc@#1}}}%
3327         {}%
3328         {\ifx\bbl@xenohyph\@undefined
3329           \let\bbl@xenohyph\bbl@xenohyph@d
3330           \ifx\AtBeginDocument\@notprerr
3331             \expandafter\@secondoftwo  % to execute right now
3332           \fi
3333           \AtBeginDocument{%
3334             \bbl@patchfont{\bbl@xenohyph}%
3335             \expandafter\selectlanguage\expandafter{\languagename}}%
3336         \fi}}%
3337   \fi
```

```
3338     \bbl@csarg\bbl@toglobal{lsys@#1}}
3339 \def\bbl@xenohyph@d{%
3340     \bbl@ifset{bbl@prehc@\languagename}%
3341       {\ifnum\hyphenchar\font=\defaulthyphenchar
3342         \iffontchar\font\bbl@cl{prehc}\relax
3343           \hyphenchar\font\bbl@cl{prehc}\relax
3344       \else\iffontchar\font"200B
3345         \hyphenchar\font"200B
3346       \else
3347         \bbl@warning
3348           {Neither 0 nor ZERO WIDTH SPACE are available\\%
3349            in the current font, and therefore the hyphen\\%
3350            will be printed. Try changing the fontspec's\\%
3351            'HyphenChar' to another value, but be aware\\%
3352            this setting is not safe (see the manual)}%
3353         \hyphenchar\font\defaulthyphenchar
3354       \fi\fi
3355     \fi}%
3356     {\hyphenchar\font\defaulthyphenchar}}
3357   % \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).

```
3358 \def\bbl@load@info#1{%
3359   \def\BabelBeforeIni##1##2{%
3360     \begingroup
3361       \bbl@read@ini{##1}0%
3362       \endinput          % babel- .tex may contain onlypreamble's
3363     \endgroup}%           boxed, to avoid extra spaces:
3364   {\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.

```
3365 \def\bbl@setdigits#1#2#3#4#5{%
3366   \bbl@exp{%
3367     \def\<\languagename digits>####1{%        ie, \langdigits
3368       \<bbl@digits@\languagename>####1\\\@nil}%
3369     \let\<bbl@cntr@digits@\languagename>\<\languagename digits>%
3370     \def\<\languagename counter>####1{%       ie, \langcounter
3371       \\\expandafter\<bbl@counter@\languagename>%
3372       \\\csname c@####1\endcsname}%
3373     \def\<bbl@counter@\languagename>####1{% ie, \bbl@counter@lang
3374       \\\expandafter\<bbl@digits@\languagename>%
3375       \\\number####1\\\@nil}}%
3376 \def\bbl@tempa##1##2##3##4##5{%
3377   \bbl@exp{%    Wow, quite a lot of hashes! :-(
3378     \def\<bbl@digits@\languagename>########1{%
3379       \\\ifx########1\\\@nil              % ie, \bbl@digits@lang
3380       \\\else
3381         \\\ifx0########1#1%
3382         \\\else\\\ifx1########1#2%
3383         \\\else\\\ifx2########1#3%
3384         \\\else\\\ifx3########1#4%
3385         \\\else\\\ifx4########1#5%
3386         \\\else\\\ifx5########1##1%
3387         \\\else\\\ifx6########1##2%
3388         \\\else\\\ifx7########1##3%
3389         \\\else\\\ifx8########1##4%
3390         \\\else\\\ifx9########1##5%
3391         \\\else########1%
3392         \\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi
```

```
3393        \\\expandafter\<bbl@digits@\languagename>%
3394      \\\fi}}}%
3395    \bbl@tempa}
```

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

```
3396 \def\bbl@buildifcase#1 {% Returns \bbl@tempa, requires \toks@={}
3397   \ifx\\#1%                % \\ before, in case #1 is multiletter
3398     \bbl@exp{%
3399       \def\\\bbl@tempa####1{%
3400         \<ifcase>####1\space\the\toks@\<else>\\\@ctrerr\<fi>}}%
3401   \else
3402     \toks@\expandafter{\the\toks@\or #1}%
3403     \expandafter\bbl@buildifcase
3404   \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).

```
3405 \newcommand\localenumeral[2]{\bbl@cs{cntr@#1@\languagename}{#2}}
3406 \def\bbl@localecntr#1#2{\localenumeral{#2}{#1}}
3407 \newcommand\localecounter[2]{%
3408   \expandafter\bbl@localecntr
3409   \expandafter{\number\csname c@#2\endcsname}{#1}}
3410 \def\bbl@alphnumeral#1#2{%
3411   \expandafter\bbl@alphnumeral@i\number#2 76543210\@@{#1}}
3412 \def\bbl@alphnumeral@i#1#2#3#4#5#6#7#8\@@#9{%
3413   \ifcase\@car#8\@nil\or    % Currenty <10000, but prepared for bigger
3414     \bbl@alphnumeral@ii{#9}000000#1\or
3415     \bbl@alphnumeral@ii{#9}00000#1#2\or
3416     \bbl@alphnumeral@ii{#9}0000#1#2#3\or
3417     \bbl@alphnumeral@ii{#9}000#1#2#3#4\else
3418     \bbl@alphnum@invalid{>9999}%
3419   \fi}
3420 \def\bbl@alphnumeral@ii#1#2#3#4#5#6#7#8{%
3421   \bbl@ifunset{bbl@cntr@#1.F.\number#5#6#7#8@\languagename}%
3422     {\bbl@cs{cntr@#1.4@\languagename}#5%
3423      \bbl@cs{cntr@#1.3@\languagename}#6%
3424      \bbl@cs{cntr@#1.2@\languagename}#7%
3425      \bbl@cs{cntr@#1.1@\languagename}#8%
3426      \ifnum#6#7#8>\z@ % TODO. An ad hoc rule for Greek. Ugly.
3427        \bbl@ifunset{bbl@cntr@#1.S.321@\languagename}{}%
3428          {\bbl@cs{cntr@#1.S.321@\languagename}}%
3429      \fi}%
3430     {\bbl@cs{cntr@#1.F.\number#5#6#7#8@\languagename}}}
3431 \def\bbl@alphnum@invalid#1{%
3432   \bbl@error{Alphabetic numeral too large (#1)}%
3433     {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.

```
3434 \def\bbl@localeinfo#1#2{%
3435   \bbl@ifunset{bbl@info@#2}{#1}%
3436     {\bbl@ifunset{bbl@\csname bbl@info@#2\endcsname @\languagename}{#1}%
3437       {\bbl@cs{\csname bbl@info@#2\endcsname @\languagename}}}}
3438 \newcommand\localeinfo[1]{%
3439   \ifx*#1\@empty   % TODO. A bit hackish to make it expandable.
3440     \bbl@afterelse\bbl@localeinfo{}%
3441   \else
3442     \bbl@localeinfo
3443       {\bbl@error{I've found no info for the current locale.\\%
3444                 The corresponding ini file has not been loaded\\%
3445                 Perhaps it doesn't exist}%
```

```
3446                    {See the manual for details.}}%
3447        {#1}%
3448    \fi}
3449 % \@namedef{bbl@info@name.locale}{lcname}
3450 \@namedef{bbl@info@tag.ini}{lini}
3451 \@namedef{bbl@info@name.english}{elname}
3452 \@namedef{bbl@info@name.opentype}{lname}
3453 \@namedef{bbl@info@tag.bcp47}{tbcp}
3454 \@namedef{bbl@info@language.tag.bcp47}{lbcp}
3455 \@namedef{bbl@info@tag.opentype}{lotf}
3456 \@namedef{bbl@info@script.name}{esname}
3457 \@namedef{bbl@info@script.name.opentype}{sname}
3458 \@namedef{bbl@info@script.tag.bcp47}{sbcp}
3459 \@namedef{bbl@info@script.tag.opentype}{sotf}
3460 \@namedef{bbl@info@region.tag.bcp47}{rbcp}
3461 \@namedef{bbl@info@variant.tag.bcp47}{vbcp}
3462 % Extensions are dealt with in a special way
3463 % Now, an internal \LaTeX{} macro:
3464 \providecommand\BCPdata[1]{\localeinfo*{#1.tag.bcp47}}
```

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

```
3465 ⟨⟨*More package options⟩⟩ ≡
3466 \DeclareOption{ensureinfo=off}{}
3467 ⟨⟨/More package options⟩⟩
3468 %
3469 \let\bbl@ensureinfo\@gobble
3470 \newcommand\BabelEnsureInfo{%
3471    \ifx\InputIfFileExists\@undefined\else
3472      \def\bbl@ensureinfo##1{%
3473        \bbl@ifunset{bbl@lname@##1}{\bbl@load@info{##1}}{}}%
3474    \fi
3475    \bbl@foreach\bbl@loaded{{%
3476      \def\languagename{##1}%
3477      \bbl@ensureinfo{##1}}}}
3478 \@ifpackagewith{babel}{ensureinfo=off}{}%
3479    {\AtEndOfPackage{% Test for plain.
3480      \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.

```
3481 \newcommand\getlocaleproperty{%
3482    \@ifstar\bbl@getproperty@s\bbl@getproperty@x}
3483 \def\bbl@getproperty@s#1#2#3{%
3484    \let#1\relax
3485    \def\bbl@elt##1##2##3{%
3486      \bbl@ifsamestring{##1/##2}{#3}%
3487        {\providecommand#1{##3}%
3488         \def\bbl@elt####1####2####3{}}%
3489        {}}%
3490    \bbl@cs{inidata@#2}}%
3491 \def\bbl@getproperty@x#1#2#3{%
3492    \bbl@getproperty@s{#1}{#2}{#3}%
3493    \ifx#1\relax
3494      \bbl@error
3495        {Unknown key for locale '#2':\\%
3496         #3\\%
3497         \string#1 will be set to \relax}%
3498        {Perhaps you misspelled it.}%
3499    \fi}
3500 \let\bbl@ini@loaded\@empty
3501 \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.

```
3502 \newcommand\babeladjust[1]{%  TODO. Error handling.
3503   \bbl@forkv{#1}{%
3504     \bbl@ifunset{bbl@ADJ@##1@##2}%
3505       {\bbl@cs{ADJ@##1}{##2}}%
3506       {\bbl@cs{ADJ@##1@##2}}}}
3507 %
3508 \def\bbl@adjust@lua#1#2{%
3509   \ifvmode
3510     \ifnum\currentgrouplevel=\z@
3511       \directlua{ Babel.#2 }%
3512       \expandafter\expandafter\expandafter\@gobble
3513     \fi
3514   \fi
3515   {\bbl@error   % The error is gobbled if everything went ok.
3516     {Currently, #1 related features can be adjusted only\\%
3517      in the main vertical list.}%
3518     {Maybe things change in the future, but this is what it is.}}}
3519 \@namedef{bbl@ADJ@bidi.mirroring@on}{%
3520   \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
3521 \@namedef{bbl@ADJ@bidi.mirroring@off}{%
3522   \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
3523 \@namedef{bbl@ADJ@bidi.text@on}{%
3524   \bbl@adjust@lua{bidi}{bidi_enabled=true}}
3525 \@namedef{bbl@ADJ@bidi.text@off}{%
3526   \bbl@adjust@lua{bidi}{bidi_enabled=false}}
3527 \@namedef{bbl@ADJ@bidi.mapdigits@on}{%
3528   \bbl@adjust@lua{bidi}{digits_mapped=true}}
3529 \@namedef{bbl@ADJ@bidi.mapdigits@off}{%
3530   \bbl@adjust@lua{bidi}{digits_mapped=false}}
3531 %
3532 \@namedef{bbl@ADJ@linebreak.sea@on}{%
3533   \bbl@adjust@lua{linebreak}{sea_enabled=true}}
3534 \@namedef{bbl@ADJ@linebreak.sea@off}{%
3535   \bbl@adjust@lua{linebreak}{sea_enabled=false}}
3536 \@namedef{bbl@ADJ@linebreak.cjk@on}{%
3537   \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
3538 \@namedef{bbl@ADJ@linebreak.cjk@off}{%
3539   \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
3540 \@namedef{bbl@ADJ@justify.arabic@on}{%
3541   \bbl@adjust@lua{linebreak}{arabic.justify_enabled=true}}
3542 \@namedef{bbl@ADJ@justify.arabic@off}{%
3543   \bbl@adjust@lua{linebreak}{arabic.justify_enabled=false}}
3544 %
3545 \def\bbl@adjust@layout#1{%
3546   \ifvmode
3547     #1%
3548     \expandafter\@gobble
3549   \fi
3550   {\bbl@error   % The error is gobbled if everything went ok.
3551     {Currently, layout related features can be adjusted only\\%
3552      in vertical mode.}%
3553     {Maybe things change in the future, but this is what it is.}}}
3554 \@namedef{bbl@ADJ@layout.tabular@on}{%
3555   \bbl@adjust@layout{\let\@tabular\bbl@NL@@tabular}}
3556 \@namedef{bbl@ADJ@layout.tabular@off}{%
3557   \bbl@adjust@layout{\let\@tabular\bbl@OL@@tabular}}
3558 \@namedef{bbl@ADJ@layout.lists@on}{%
3559   \bbl@adjust@layout{\let\list\bbl@NL@list}}
3560 \@namedef{bbl@ADJ@layout.lists@off}{%
3561   \bbl@adjust@layout{\let\list\bbl@OL@list}}
```

```
3562 \@namedef{bbl@ADJ@hyphenation.extra@on}{%
3563   \bbl@activateposthyphen}
3564 %
3565 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
3566   \bbl@bcpallowedtrue}
3567 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
3568   \bbl@bcpallowedfalse}
3569 \@namedef{bbl@ADJ@autoload.bcp47.prefix}#1{%
3570   \def\bbl@bcp@prefix{#1}}
3571 \def\bbl@bcp@prefix{bcp47-}
3572 \@namedef{bbl@ADJ@autoload.options}#1{%
3573   \def\bbl@autoload@options{#1}}
3574 \let\bbl@autoload@bcpoptions\@empty
3575 \@namedef{bbl@ADJ@autoload.bcp47.options}#1{%
3576   \def\bbl@autoload@bcpoptions{#1}}
3577 \newif\ifbbl@bcptoname
3578 \@namedef{bbl@ADJ@bcp47.toname@on}{%
3579   \bbl@bcptonametrue
3580   \BabelEnsureInfo}
3581 \@namedef{bbl@ADJ@bcp47.toname@off}{%
3582   \bbl@bcptonamefalse}
3583 \@namedef{bbl@ADJ@prehyphenation.disable@nohyphenation}{%
3584   \directlua{ Babel.ignore_pre_char = function(node)
3585     return (node.lang == \the\csname l@nohyphenation\endcsname)
3586   end }}
3587 \@namedef{bbl@ADJ@prehyphenation.disable@off}{%
3588   \directlua{ Babel.ignore_pre_char = function(node)
3589     return false
3590   end }}
3591 \@namedef{bbl@ADJ@select.write@shift}{%
3592   \let\bbl@restorelastskip\relax
3593   \def\bbl@savelastskip{%
3594     \let\bbl@restorelastskip\relax
3595     \ifvmode
3596       \ifdim\lastskip=\z@
3597         \let\bbl@restorelastskip\nobreak
3598       \else
3599         \bbl@exp{%
3600           \def\\\bbl@restorelastskip{%
3601             \skip@=\the\lastskip
3602             \\\nobreak \vskip-\skip@ \vskip\skip@}}%
3603       \fi
3604   \fi}}
3605 \@namedef{bbl@ADJ@select.write@keep}{%
3606   \let\bbl@restorelastskip\relax
3607   \let\bbl@savelastskip\relax}
3608 \@namedef{bbl@ADJ@select.write@omit}{%
3609   \let\bbl@restorelastskip\relax
3610   \def\bbl@savelastskip##1\bbl@restorelastskip{}}
```

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

```
3611 \ifx\directlua\@undefined\else
3612   \ifx\bbl@luapatterns\@undefined
3613     \input luababel.def
3614   \fi
3615 \fi
```

Continue with LaTeX.

```
3616 ⟨/package | core⟩
3617 ⟨*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.

```
3618 ⟨⟨*More package options⟩⟩ ≡
3619 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
3620 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
3621 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
3622 \DeclareOption{safe=refbib}{\def\bbl@opt@safe{BR}}
3623 \DeclareOption{safe=bibref}{\def\bbl@opt@safe{BR}}
3624 ⟨⟨/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.

```
3625 \bbl@trace{Cross referencing macros}
3626 \ifx\bbl@opt@safe\@empty\else % ie, if 'ref' and/or 'bib'
3627   \def\@newl@bel#1#2#3{%
3628     {\@safe@activestrue
3629      \bbl@ifunset{#1@#2}%
3630        \relax
3631        {\gdef\@multiplelabels{%
3632           \@latex@warning@no@line{There were multiply-defined labels}}%
3633         \@latex@warning@no@line{Label `#2' multiply defined}}%
3634     \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.

```
3635   \CheckCommand*\@testdef[3]{%
3636     \def\reserved@a{#3}%
3637     \expandafter\ifx\csname#1@#2\endcsname\reserved@a
3638     \else
3639       \@tempswatrue
3640     \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.

```
3641   \def\@testdef#1#2#3{%  TODO. With @samestring?
3642     \@safe@activestrue
3643     \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
3644     \def\bbl@tempb{#3}%
3645     \@safe@activesfalse
3646     \ifx\bbl@tempa\relax
3647     \else
3648       \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
3649     \fi
3650     \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
3651     \ifx\bbl@tempa\bbl@tempb
3652     \else
3653       \@tempswatrue
3654     \fi}
3655 \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.

```
3656 \bbl@xin@{R}\bbl@opt@safe
3657 \ifin@
3658   \edef\bbl@tempc{\expandafter\string\csname ref code\endcsname}%
3659   \bbl@xin@{\expandafter\strip@prefix\meaning\bbl@tempc}%
3660     {\expandafter\strip@prefix\meaning\ref}%
3661   \ifin@
3662     \bbl@redefine\@kernel@ref#1{%
3663       \@safe@activestrue\org@@kernel@ref{#1}\@safe@activesfalse}
3664     \bbl@redefine\@kernel@pageref#1{%
3665       \@safe@activestrue\org@@kernel@pageref{#1}\@safe@activesfalse}
3666     \bbl@redefine\@kernel@sref#1{%
3667       \@safe@activestrue\org@@kernel@sref{#1}\@safe@activesfalse}
3668     \bbl@redefine\@kernel@spageref#1{%
3669       \@safe@activestrue\org@@kernel@spageref{#1}\@safe@activesfalse}
3670   \else
3671     \bbl@redefinerobust\ref#1{%
3672       \@safe@activestrue\org@ref{#1}\@safe@activesfalse}
3673     \bbl@redefinerobust\pageref#1{%
3674       \@safe@activestrue\org@pageref{#1}\@safe@activesfalse}
3675   \fi
3676 \else
3677   \let\org@ref\ref
3678   \let\org@pageref\pageref
3679 \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.

```
3680 \bbl@xin@{B}\bbl@opt@safe
3681 \ifin@
3682   \bbl@redefine\@citex[#1]#2{%
3683     \@safe@activestrue\edef\@tempa{#2}\@safe@activesfalse
3684     \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.

```
3685   \AtBeginDocument{%
3686     \@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.)

```
3687     \def\@citex[#1][#2]#3{%
3688       \@safe@activestrue\edef\@tempa{#3}\@safe@activesfalse
3689       \org@@citex[#1][#2]{\@tempa}}%
3690     }{}}
```

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

```
3691   \AtBeginDocument{%
3692     \@ifpackageloaded{cite}{%
3693       \def\@citex[#1]#2{%
3694         \@safe@activestrue\org@@citex[#1]{#2}\@safe@activesfalse}%
3695       }{}}
```

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

```
3696 \bbl@redefine\nocite#1{%
3697   \@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.

```
3698 \bbl@redefine\bibcite{%
3699     \bbl@cite@choice
3700     \bibcite}
```

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

```
3701 \def\bbl@bibcite#1#2{%
3702     \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.

```
3703 \def\bbl@cite@choice{%
3704     \global\let\bibcite\bbl@bibcite
3705     \@ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{}%
3706     \@ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{}%
3707     \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.

```
3708 \AtBeginDocument{\bbl@cite@choice}
```

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

```
3709 \bbl@redefine\@bibitem#1{%
3710     \@safe@activestrue\org@@bibitem{#1}\@safe@activesfalse}
3711 \else
3712 \let\org@nocite\nocite
3713 \let\org@@citex\@citex
3714 \let\org@bibcite\bibcite
3715 \let\org@@bibitem\@bibitem
3716 \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.

```
3717 \bbl@trace{Marks}
3718 \IfBabelLayout{sectioning}
3719  {\ifx\bbl@opt@headfoot\@nnil
3720     \g@addto@macro\@resetactivechars{%
3721       \set@typeset@protect
3722       \expandafter\select@language@x\expandafter{\bbl@main@language}%
3723       \let\protect\noexpand
3724       \ifcase\bbl@bidimode\else % Only with bidi. See also above
3725         \edef\thepage{%
3726           \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
3727       \fi}%
3728   \fi}
3729  {\ifbbl@single\else
3730     \bbl@ifunset{markright }\bbl@redefine\bbl@redefinerobust
3731     \markright#1{%
3732       \bbl@ifblank{#1}%
```

```
3733              {\org@markright{}}%
3734              {\toks@{#1}%
3735               \bbl@exp{%
3736                  \\\org@markright{\\\protect\\\foreignlanguage{\languagename}%
3737                    {\\\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.)

```
3738      \ifx\@mkboth\markboth
3739        \def\bbl@tempc{\let\@mkboth\markboth}
3740      \else
3741        \def\bbl@tempc{}
3742      \fi
3743      \bbl@ifunset{markboth }\bbl@redefine\bbl@redefinerobust
3744      \markboth#1#2{%
3745        \protected@edef\bbl@tempb##1{%
3746          \protect\foreignlanguage
3747          {\languagename}{\protect\bbl@restore@actives##1}}%
3748        \bbl@ifblank{#1}%
3749          {\toks@{}}%
3750          {\toks@\expandafter{\bbl@tempb{#1}}}%
3751        \bbl@ifblank{#2}%
3752          {\@temptokena{}}%
3753          {\@temptokena\expandafter{\bbl@tempb{#2}}}%
3754        \bbl@exp{\\\org@markboth{\the\toks@}{\the\@temptokena}}}
3755        \bbl@tempc
3756      \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.

```
3757 \bbl@trace{Preventing clashes with other packages}
3758 \ifx\org@ref\@undefined\else
3759   \bbl@xin@{R}\bbl@opt@safe
3760   \ifin@
3761     \AtBeginDocument{%
3762       \@ifpackageloaded{ifthen}{%
3763         \bbl@redefine@long\ifthenelse#1#2#3{%
3764           \let\bbl@temp@pref\pageref
3765           \let\pageref\org@pageref
3766           \let\bbl@temp@ref\ref
3767           \let\ref\org@ref
```

```
3768            \@safe@activestrue
3769            \org@ifthenelse{#1}%
3770              {\let\pageref\bbl@temp@pref
3771               \let\ref\bbl@temp@ref
3772               \@safe@activesfalse
3773               #2}%
3774              {\let\pageref\bbl@temp@pref
3775               \let\ref\bbl@temp@ref
3776               \@safe@activesfalse
3777               #3}%
3778            }%
3779          }{}%
3780       }
3781 \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.

```
3782   \AtBeginDocument{%
3783     \@ifpackageloaded{varioref}{%
3784       \bbl@redefine\@@vpageref#1[#2]#3{%
3785         \@safe@activestrue
3786         \org@@@vpageref{#1}[#2]{#3}%
3787         \@safe@activesfalse}%
3788       \bbl@redefine\vrefpagenum#1#2{%
3789         \@safe@activestrue
3790         \org@vrefpagenum{#1}{#2}%
3791         \@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.

```
3792       \expandafter\def\csname Ref \endcsname#1{%
3793         \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
3794     }{}%
3795   }
3796 \fi
```

8.3.3 hhline

\hhline   Delaying the activation of the shorthand characters has introduced a problem with the hhline
package. The reason is that it uses the ':' character which is made active by the french support in
babel. Therefore we need to *reload* the package when the ':' is an active character. Note that this
happens *after* the category code of the @-sign has been changed to other, so we need to temporarily
change it to letter again.

```
3797 \AtEndOfPackage{%
3798   \AtBeginDocument{%
3799     \@ifpackageloaded{hhline}%
3800       {\expandafter\ifx\csname normal@char\string:\endcsname\relax
3801         \else
3802           \makeatletter
3803           \def\@currname{hhline}\input{hhline.sty}\makeatother
3804         \fi}%
3805       {}}}
```

\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.

```
3806 \def\substitutefontfamily#1#2#3{%
3807   \lowercase{\immediate\openout15=#1#2.fd\relax}%
3808   \immediate\write15{%
3809     \string\ProvidesFile{#1#2.fd}%
3810     [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
3811      \space generated font description file]^^J
3812     \string\DeclareFontFamily{#1}{#2}{}^^J
3813     \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{}^^J
3814     \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{}^^J
3815     \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{}^^J
3816     \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{}^^J
3817     \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{}^^J
3818     \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{}^^J
3819     \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{}^^J
3820     \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{}^^J
3821     }%
3822   \closeout15
3823   }
3824 \@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`

```
3825 \bbl@trace{Encoding and fonts}
3826 \newcommand\BabelNonASCII{LGR,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU}
3827 \newcommand\BabelNonText{TS1,T3,TS3}
3828 \let\org@TeX\TeX
3829 \let\org@LaTeX\LaTeX
3830 \let\ensureascii\@firstofone
3831 \AtBeginDocument{%
3832   \def\@elt#1{,#1,}%
3833   \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3834   \let\@elt\relax
3835   \let\bbl@tempb\@empty
3836   \def\bbl@tempc{OT1}%
3837   \bbl@foreach\BabelNonASCII{% LGR loaded in a non-standard way
3838     \bbl@ifunset{T@#1}{}{\def\bbl@tempb{#1}}}%
3839   \bbl@foreach\bbl@tempa{%
3840     \bbl@xin@{#1}{\BabelNonASCII}%
3841     \ifin@
3842       \def\bbl@tempb{#1}% Store last non-ascii
3843     \else\bbl@xin@{#1}{\BabelNonText}% Pass
3844       \ifin@\else
3845         \def\bbl@tempc{#1}% Store last ascii
3846       \fi
3847     \fi}%
3848   \ifx\bbl@tempb\@empty\else
3849     \bbl@xin@{,\cf@encoding,}{,\BabelNonASCII,\BabelNonText,}%
3850     \ifin@\else
3851       \edef\bbl@tempc{\cf@encoding}% The default if ascii wins
3852     \fi
3853     \edef\ensureascii#1{%
3854       {\noexpand\fontencoding{\bbl@tempc}\noexpand\selectfont#1}}%
3855     \DeclareTextCommandDefault{\TeX}{\ensureascii{\org@TeX}}%
3856     \DeclareTextCommandDefault{\LaTeX}{\ensureascii{\org@LaTeX}}%
3857   \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.

```
3858 \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.

```
3859 \AtBeginDocument{%
3860   \@ifpackageloaded{fontspec}%
3861     {\xdef\latinencoding{%
3862       \ifx\UTFencname\@undefined
3863         EU\ifcase\bbl@engine\or2\or1\fi
3864       \else
3865         \UTFencname
3866       \fi}}%
3867   {\gdef\latinencoding{OT1}%
3868    \ifx\cf@encoding\bbl@t@one
3869      \xdef\latinencoding{\bbl@t@one}%
3870    \else
3871    \def\@elt#1{,#1,}%
3872    \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3873    \let\@elt\relax
3874    \bbl@xin@{,T1,}\bbl@tempa
3875    \ifin@
3876      \xdef\latinencoding{\bbl@t@one}%
3877    \fi
3878   \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.

```
3879 \DeclareRobustCommand{\latintext}{%
3880   \fontencoding{\latinencoding}\selectfont
3881   \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.

```
3882 \ifx\@undefined\DeclareTextFontCommand
3883   \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
3884 \else
3885   \DeclareTextFontCommand{\textlatin}{\latintext}
3886 \fi
```

For several functions, we need to execute some code with `\selectfont`. With LaTeX 2021-06-01, there is a hook for this purpose, but in older versions the LaTeX command is patched (the latter solution will be eventually removed).

```
3887 \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.

```
3888 \bbl@trace{Loading basic (internal) bidi support}
3889 \ifodd\bbl@engine
3890 \else % TODO. Move to txtbabel
3891   \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
3892     \bbl@error
3893       {The bidi method 'basic' is available only in\\%
3894        luatex. I'll continue with 'bidi=default', so\\%
3895        expect wrong results}%
3896       {See the manual for further details.}%
3897     \let\bbl@beforeforeign\leavevmode
3898     \AtEndOfPackage{%
3899       \EnableBabelHook{babel-bidi}%
3900       \bbl@xebidipar}
3901   \fi\fi
3902   \def\bbl@loadxebidi#1{%
3903     \ifx\RTLfootnotetext\@undefined
3904       \AtEndOfPackage{%
3905         \EnableBabelHook{babel-bidi}%
3906         \ifx\fontspec\@undefined
3907           \bbl@loadfontspec % bidi needs fontspec
3908         \fi
3909         \usepackage#1{bidi}}%
3910     \fi}
3911   \ifnum\bbl@bidimode>200
3912     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
3913       \bbl@tentative{bidi=bidi}
3914       \bbl@loadxebidi{}
3915     \or
3916       \bbl@loadxebidi{[rldocument]}
3917     \or
3918       \bbl@loadxebidi{}
3919     \fi
3920   \fi
3921 \fi
3922 % TODO? Separate:
3923 \ifnum\bbl@bidimode=\@ne
3924   \let\bbl@beforeforeign\leavevmode
3925   \ifodd\bbl@engine
3926     \newattribute\bbl@attr@dir
3927     \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
3928     \bbl@exp{\output{\bodydir\pagedir\the\output}}
3929   \fi
3930   \AtEndOfPackage{%
3931     \EnableBabelHook{babel-bidi}%
3932     \ifodd\bbl@engine\else
3933       \bbl@xebidipar
3934     \fi}
3935 \fi
```

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

```
3936 \bbl@trace{Macros to switch the text direction}
3937 \def\bbl@alscripts{,Arabic,Syriac,Thaana,}
```

```
3938 \def\bbl@rscripts{% TODO. Base on codes ??
3939   ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
3940   Old Hungarian,Old Hungarian,Lydian,Mandaean,Manichaean,%
3941   Manichaean,Meroitic Cursive,Meroitic,Old North Arabian,%
3942   Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
3943   Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
3944   Old South Arabian,}%
3945 \def\bbl@provide@dirs#1{%
3946   \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
3947   \ifin@
3948     \global\bbl@csarg\chardef{wdir@#1}\@ne
3949     \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
3950     \ifin@
3951       \global\bbl@csarg\chardef{wdir@#1}\tw@  % useless in xetex
3952     \fi
3953   \else
3954     \global\bbl@csarg\chardef{wdir@#1}\z@
3955   \fi
3956   \ifodd\bbl@engine
3957     \bbl@csarg\ifcase{wdir@#1}%
3958       \directlua{ Babel.locale_props[\the\localeid].textdir = 'l' }%
3959     \or
3960       \directlua{ Babel.locale_props[\the\localeid].textdir = 'r' }%
3961     \or
3962       \directlua{ Babel.locale_props[\the\localeid].textdir = 'al' }%
3963     \fi
3964   \fi}
3965 \def\bbl@switchdir{%
3966   \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
3967   \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
3968   \bbl@exp{\\\bbl@setdirs\bbl@cl{wdir}}}
3969 \def\bbl@setdirs#1{% TODO - math
3970   \ifcase\bbl@select@type % TODO - strictly, not the right test
3971     \bbl@bodydir{#1}%
3972     \bbl@pardir{#1}%
3973   \fi
3974   \bbl@textdir{#1}}
3975 % TODO. Only if \bbl@bidimode > 0?:
3976 \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
3977 \DisableBabelHook{babel-bidi}
```

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

```
3978 \ifodd\bbl@engine  % luatex=1
3979 \else % pdftex=0, xetex=2
3980   \newcount\bbl@dirlevel
3981   \chardef\bbl@thetextdir\z@
3982   \chardef\bbl@thepardir\z@
3983   \def\bbl@textdir#1{%
3984     \ifcase#1\relax
3985       \chardef\bbl@thetextdir\z@
3986       \bbl@textdir@i\beginL\endL
3987     \else
3988       \chardef\bbl@thetextdir\@ne
3989       \bbl@textdir@i\beginR\endR
3990     \fi}
3991   \def\bbl@textdir@i#1#2{%
3992     \ifhmode
3993       \ifnum\currentgrouplevel>\z@
3994         \ifnum\currentgrouplevel=\bbl@dirlevel
3995           \bbl@error{Multiple bidi settings inside a group}%
3996             {I'll insert a new group, but expect wrong results.}%
3997           \bgroup\aftergroup#2\aftergroup\egroup
3998         \else
```

```
3999        \ifcase\currentgrouptype\or % 0 bottom
4000          \aftergroup#2% 1 simple {}
4001        \or
4002          \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
4003        \or
4004          \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
4005        \or\or\or % vbox vtop align
4006        \or
4007          \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
4008        \or\or\or\or\or\or % output math disc insert vcent mathchoice
4009        \or
4010          \aftergroup#2% 14 \begingroup
4011        \else
4012          \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
4013        \fi
4014      \fi
4015      \bbl@dirlevel\currentgrouplevel
4016    \fi
4017    #1%
4018  \fi}
4019 \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
4020 \let\bbl@bodydir\@gobble
4021 \let\bbl@pagedir\@gobble
4022 \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).

```
4023 \def\bbl@xebidipar{%
4024   \let\bbl@xebidipar\relax
4025   \TeXXeTstate\@ne
4026   \def\bbl@xeeverypar{%
4027     \ifcase\bbl@thepardir
4028       \ifcase\bbl@thetextdir\else\beginR\fi
4029     \else
4030       {\setbox\z@\lastbox\beginR\box\z@}%
4031     \fi}%
4032   \let\bbl@severypar\everypar
4033   \newtoks\everypar
4034   \everypar=\bbl@severypar
4035   \bbl@severypar{\bbl@xeeverypar\the\everypar}}
4036 \ifnum\bbl@bidimode>200
4037   \let\bbl@textdir@i\@gobbletwo
4038   \let\bbl@xebidipar\@empty
4039   \AddBabelHook{bidi}{foreign}{%
4040     \def\bbl@tempa{\def\BabelText####1}%
4041     \ifcase\bbl@thetextdir
4042       \expandafter\bbl@tempa\expandafter{\BabelText{\LR{##1}}}%
4043     \else
4044       \expandafter\bbl@tempa\expandafter{\BabelText{\RL{##1}}}%
4045     \fi}
4046   \def\bbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}
4047 \fi
4048 \fi
```

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

```
4049 \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}
4050 \AtBeginDocument{%
4051   \ifx\pdfstringdefDisableCommands\@undefined\else
4052     \ifx\pdfstringdefDisableCommands\relax\else
4053       \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
4054     \fi
4055   \fi}
```

144

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.

```
4056 \bbl@trace{Local Language Configuration}
4057 \ifx\loadlocalcfg\@undefined
4058   \@ifpackagewith{babel}{noconfigs}%
4059     {\let\loadlocalcfg\@gobble}%
4060     {\def\loadlocalcfg#1{%
4061       \InputIfFileExists{#1.cfg}%
4062         {\typeout{*************************************^^J%
4063                       * Local config file #1.cfg used^^J%
4064                       *}}%
4065       \@empty}}
4066 \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).

```
4067 \bbl@trace{Language options}
4068 \let\bbl@afterlang\relax
4069 \let\BabelModifiers\relax
4070 \let\bbl@loaded\@empty
4071 \def\bbl@load@language#1{%
4072   \InputIfFileExists{#1.ldf}%
4073     {\edef\bbl@loaded{\CurrentOption
4074       \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}%
4075     \expandafter\let\expandafter\bbl@afterlang
4076       \csname\CurrentOption.ldf-h@@k\endcsname
4077     \expandafter\let\expandafter\BabelModifiers
4078       \csname bbl@mod@\CurrentOption\endcsname}%
4079     {\bbl@error{%
4080       Unknown option '\CurrentOption'. Either you misspelled it\\%
4081       or the language definition file \CurrentOption.ldf was not found}{%
4082       Valid options are, among others: shorthands=, KeepShorthandsActive,\\%
4083       activeacute, activegrave, noconfigs, safe=, main=, math=\\%
4084       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.

```
4085 \def\bbl@try@load@lang#1#2#3{%
4086   \IfFileExists{\CurrentOption.ldf}%
4087     {\bbl@load@language{\CurrentOption}}%
4088     {#1\bbl@load@language{#2}#3}}
4089 %
4090 \DeclareOption{hebrew}{%
4091   \input{rlbabel.def}%
4092   \bbl@load@language{hebrew}}
4093 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}{}}
4094 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}{}}
4095 \DeclareOption{nynorsk}{\bbl@try@load@lang{}{norsk}{}}
4096 \DeclareOption{polutonikogreek}{%
4097   \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}
4098 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}{}}
4099 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}{}}
4100 \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.

```
4101 \ifx\bbl@opt@config\@nnil
4102   \@ifpackagewith{babel}{noconfigs}{}%
4103     {\InputIfFileExists{bblopts.cfg}%
4104       {\typeout{***********************************^^J%
4105               * Local config file bblopts.cfg used^^J%
4106               *}}%
4107     {}}%
4108 \else
4109   \InputIfFileExists{\bbl@opt@config.cfg}%
4110     {\typeout{***********************************^^J%
4111               * Local config file \bbl@opt@config.cfg used^^J%
4112               *}}%
4113     {\bbl@error{%
4114       Local config file '\bbl@opt@config.cfg' not found}{%
4115       Perhaps you misspelled it.}}%
4116 \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.

```
4117 \ifx\bbl@opt@main\@nnil
4118   \ifnum\bbl@iniflag>\z@  % if all ldf's: set implicitly, no main pass
4119     \let\bbl@tempb\@empty
4120     \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}%
4121     \bbl@foreach\bbl@tempa{\edef\bbl@tempb{#1,\bbl@tempb}}%
4122     \bbl@foreach\bbl@tempb{%    \bbl@tempb is a reversed list
4123       \ifx\bbl@opt@main\@nnil % ie, if not yet assigned
4124         \ifodd\bbl@iniflag % = *=
4125           \IfFileExists{babel-#1.tex}{\def\bbl@opt@main{#1}}{}%
4126         \else % n +=
4127           \IfFileExists{#1.ldf}{\def\bbl@opt@main{#1}}{}%
4128         \fi
4129       \fi}%
4130   \fi
4131 \else
4132   \bbl@info{Main language set with 'main='. Except if you have\\%
4133            problems, prefer the default mechanism for setting\\%
4134            the main language. Reported}
4135 \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).

```
4136 \ifx\bbl@opt@main\@nnil\else
4137   \bbl@csarg\let{loadmain\expandafter}\csname ds@\bbl@opt@main\endcsname
4138   \expandafter\let\csname ds@\bbl@opt@main\endcsname\relax
4139 \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.

```
4140 \bbl@foreach\bbl@language@opts{%
4141   \def\bbl@tempa{#1}%
4142   \ifx\bbl@tempa\bbl@opt@main\else
4143     \ifnum\bbl@iniflag<\tw@    % 0 ø (other = ldf)
4144       \bbl@ifunset{ds@#1}%
4145         {\DeclareOption{#1}{\bbl@load@language{#1}}}%
```

```
4146          {}%
4147      \else                        % + * (other = ini)
4148        \DeclareOption{#1}{%
4149          \bbl@ldfinit
4150          \babelprovide[import]{#1}%
4151          \bbl@afterldf{}}%
4152      \fi
4153    \fi}
4154 \bbl@foreach\@classoptionslist{%
4155    \def\bbl@tempa{#1}%
4156    \ifx\bbl@tempa\bbl@opt@main\else
4157      \ifnum\bbl@iniflag<\tw@    % 0 ø (other = ldf)
4158        \bbl@ifunset{ds@#1}%
4159          {\IfFileExists{#1.ldf}%
4160            {\DeclareOption{#1}{\bbl@load@language{#1}}}%
4161            {}}%
4162          {}%
4163      \else                        % + * (other = ini)
4164        \IfFileExists{babel-#1.tex}%
4165          {\DeclareOption{#1}{%
4166            \bbl@ldfinit
4167            \babelprovide[import]{#1}%
4168            \bbl@afterldf{}}}%
4169          {}%
4170      \fi
4171    \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):

```
4172 \def\AfterBabelLanguage#1{%
4173    \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang}{}}
4174 \DeclareOption*{}
4175 \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.

```
4176 \bbl@trace{Option 'main'}
4177 \ifx\bbl@opt@main\@nnil
4178    \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}
4179    \let\bbl@tempc\@empty
4180    \bbl@for\bbl@tempb\bbl@tempa{%
4181      \bbl@xin@{,\bbl@tempb,}{,\bbl@loaded,}%
4182      \ifin@\edef\bbl@tempc{\bbl@tempb}\fi}
4183    \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
4184    \expandafter\bbl@tempa\bbl@loaded,\@nnil
4185    \ifx\bbl@tempb\bbl@tempc\else
4186      \bbl@warning{%
4187        Last declared language option is '\bbl@tempc',\\%
4188        but the last processed one was '\bbl@tempb'.\\%
4189        The main language can't be set as both a global\\%
4190        and a package option. Use 'main=\bbl@tempc' as\\%
4191        option. Reported}
4192    \fi
4193 \else
4194    \ifodd\bbl@iniflag  % case 1,3 (main is ini)
4195      \bbl@ldfinit
4196      \let\CurrentOption\bbl@opt@main
4197      \bbl@exp{%  \bbl@opt@provide = empty if *
```

147

```
4198        \\\babelprovide[\bbl@opt@provide,import,main]{\bbl@opt@main}}%
4199      \bbl@afterldf{}
4200      \DeclareOption{\bbl@opt@main}{}
4201    \else % case 0,2 (main is ldf)
4202      \ifx\bbl@loadmain\relax
4203        \DeclareOption{\bbl@opt@main}{\bbl@load@language{\bbl@opt@main}}
4204      \else
4205        \DeclareOption{\bbl@opt@main}{\bbl@loadmain}
4206      \fi
4207      \ExecuteOptions{\bbl@opt@main}
4208      \@namedef{ds@\bbl@opt@main}{}%
4209    \fi
4210    \DeclareOption*{}
4211    \ProcessOptions*
4212 \fi
4213 \def\AfterBabelLanguage{%
4214    \bbl@error
4215      {Too late for \string\AfterBabelLanguage}%
4216      {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.

```
4217 \ifx\bbl@main@language\@undefined
4218    \bbl@info{%
4219      You haven't specified a language. I'll use 'nil'\\%
4220      as the main language. Reported}
4221      \bbl@load@language{nil}
4222 \fi
4223 ⟨/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

```
4224 ⟨*kernel⟩
4225 \let\bbl@onlyswitch\@empty
4226 \input babel.def
4227 \let\bbl@onlyswitch\@undefined
4228 ⟨/kernel⟩
4229 ⟨*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.

```
4230 ⟨⟨Make sure ProvidesFile is defined⟩⟩
4231 \ProvidesFile{hyphen.cfg}[⟨⟨date⟩⟩ ⟨⟨version⟩⟩ Babel hyphens]
4232 \xdef\bbl@format{\jobname}
4233 \def\bbl@version{⟨⟨version⟩⟩}
4234 \def\bbl@date{⟨⟨date⟩⟩}
4235 \ifx\AtBeginDocument\@undefined
```

```
4236    \def\@empty{}
4237 \fi
```
4238 ⟨⟨*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.

```
4239 \def\process@line#1#2 #3 #4 {%
4240   \ifx=#1%
4241     \process@synonym{#2}%
4242   \else
4243     \process@language{#1#2}{#3}{#4}%
4244   \fi
4245   \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.

```
4246 \toks@{}
4247 \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.

```
4248 \def\process@synonym#1{%
4249   \ifnum\last@language=\m@ne
4250     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4251   \else
4252     \expandafter\chardef\csname l@#1\endcsname\last@language
4253     \wlog{\string\l@#1=\string\language\the\last@language}%
4254     \expandafter\let\csname #1hyphenmins\expandafter\endcsname
4255       \csname\languagename hyphenmins\endcsname
4256     \let\bbl@elt\relax
4257     \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}{}{}}%
4258   \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.

```
4259 \def\process@language#1#2#3{%
4260   \expandafter\addlanguage\csname l@#1\endcsname
4261   \expandafter\language\csname l@#1\endcsname
4262   \edef\languagename{#1}%
4263   \bbl@hook@everylanguage{#1}%
4264   %  > luatex
4265   \bbl@get@enc#1::\@@@
4266   \begingroup
4267     \lefthyphenmin\m@ne
4268     \bbl@hook@loadpatterns{#2}%
4269     %  > luatex
4270     \ifnum\lefthyphenmin=\m@ne
4271     \else
4272       \expandafter\xdef\csname #1hyphenmins\endcsname{%
4273         \the\lefthyphenmin\the\righthyphenmin}%
4274     \fi
4275   \endgroup
4276   \def\bbl@tempa{#3}%
4277   \ifx\bbl@tempa\@empty\else
4278     \bbl@hook@loadexceptions{#3}%
4279     %  > luatex
4280   \fi
4281   \let\bbl@elt\relax
4282   \edef\bbl@languages{%
4283     \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
4284   \ifnum\the\language=\z@
4285     \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4286       \set@hyphenmins\tw@\thr@@\relax
4287     \else
4288       \expandafter\expandafter\expandafter\set@hyphenmins
4289         \csname #1hyphenmins\endcsname
4290     \fi
4291     \the\toks@
4292     \toks@{}%
4293   \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.

```
4294 \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.

```
4295 \def\bbl@hook@everylanguage#1{}
4296 \def\bbl@hook@loadpatterns#1{\input #1\relax}
4297 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
4298 \def\bbl@hook@loadkernel#1{%
4299   \def\addlanguage{\csname newlanguage\endcsname}%
4300   \def\adddialect##1##2{%
4301     \global\chardef##1##2\relax
4302     \wlog{\string##1 = a dialect from \string\language##2}}%
4303   \def\iflanguage##1{%
4304     \expandafter\ifx\csname l@##1\endcsname\relax
4305       \@nolanerr{##1}%
4306     \else
4307       \ifnum\csname l@##1\endcsname=\language
4308         \expandafter\expandafter\expandafter\@firstoftwo
4309       \else
4310         \expandafter\expandafter\expandafter\@secondoftwo
4311       \fi
4312     \fi}%
```

```
4313    \def\providehyphenmins##1##2{%
4314      \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
4315        \@namedef{##1hyphenmins}{##2}%
4316      \fi}%
4317    \def\set@hyphenmins##1##2{%
4318      \lefthyphenmin##1\relax
4319      \righthyphenmin##2\relax}%
4320    \def\selectlanguage{%
4321      \errhelp{Selecting a language requires a package supporting it}%
4322      \errmessage{Not loaded}}%
4323    \let\foreignlanguage\selectlanguage
4324    \let\otherlanguage\selectlanguage
4325    \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4326    \def\bbl@usehooks##1##2{}% TODO. Temporary!!
4327    \def\setlocale{%
4328      \errhelp{Find an armchair, sit down and wait}%
4329      \errmessage{Not yet available}}%
4330    \let\uselocale\setlocale
4331    \let\locale\setlocale
4332    \let\selectlocale\setlocale
4333    \let\localename\setlocale
4334    \let\textlocale\setlocale
4335    \let\textlanguage\setlocale
4336    \let\languagetext\setlocale}
4337 \begingroup
4338    \def\AddBabelHook#1#2{%
4339      \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
4340        \def\next{\toks1}%
4341      \else
4342        \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname####1}%
4343      \fi
4344      \next}
4345    \ifx\directlua\@undefined
4346      \ifx\XeTeXinputencoding\@undefined\else
4347        \input xebabel.def
4348      \fi
4349    \else
4350      \input luababel.def
4351    \fi
4352    \openin1 = babel-\bbl@format.cfg
4353    \ifeof1
4354    \else
4355      \input babel-\bbl@format.cfg\relax
4356    \fi
4357    \closein1
4358 \endgroup
4359 \bbl@hook@loadkernel{switch.def}
```

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

```
4360 \openin1 = language.dat
```

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

```
4361 \def\languagename{english}%
4362 \ifeof1
4363   \message{I couldn't find the file language.dat,\space
4364           I will try the file hyphen.tex}
4365   \input hyphen.tex\relax
4366   \chardef\l@english\z@
4367 \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.

```
4368    \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.

```
4369    \loop
4370      \endlinechar\m@ne
4371      \read1 to \bbl@line
4372      \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.

```
4373      \if T\ifeof1F\fi T\relax
4374        \ifx\bbl@line\@empty\else
4375          \edef\bbl@line{\bbl@line\space\space\space}%
4376          \expandafter\process@line\bbl@line\relax
4377        \fi
4378    \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.

```
4379    \begingroup
4380      \def\bbl@elt#1#2#3#4{%
4381        \global\language=#2\relax
4382        \gdef\languagename{#1}%
4383        \def\bbl@elt##1##2##3##4{}}%
4384      \bbl@languages
4385    \endgroup
4386 \fi
4387 \closein1
```

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

```
4388 \if/\the\toks@/\else
4389    \errhelp{language.dat loads no language, only synonyms}
4390    \errmessage{Orphan language synonym}
4391 \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.

```
4392 \let\bbl@line\@undefined
4393 \let\process@line\@undefined
4394 \let\process@synonym\@undefined
4395 \let\process@language\@undefined
4396 \let\bbl@get@enc\@undefined
4397 \let\bbl@hyph@enc\@undefined
4398 \let\bbl@tempa\@undefined
4399 \let\bbl@hook@loadkernel\@undefined
4400 \let\bbl@hook@everylanguage\@undefined
4401 \let\bbl@hook@loadpatterns\@undefined
4402 \let\bbl@hook@loadexceptions\@undefined
4403 ⟨/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].

```
4404 ⟨⟨*More package options⟩⟩ ≡
```

```
4405 \chardef\bbl@bidimode\z@
4406 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4407 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
4408 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4409 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4410 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4411 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4412 ⟨⟨/More package options⟩⟩
```

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

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

```
4413 ⟨⟨∗Font selection⟩⟩ ≡
4414 \bbl@trace{Font handling with fontspec}
4415 \ifx\ExplSyntaxOn\@undefined\else
4416   \ExplSyntaxOn
4417   \catcode`\ =10
4418   \def\bbl@loadfontspec{%
4419     \usepackage{fontspec}%  TODO. Apply patch always
4420     \expandafter
4421     \def\csname msg~text~>~fontspec/language-not-exist\endcsname##1##2##3##4{%
4422       Font '\l_fontspec_fontname_tl' is using the\\%
4423       default features for language '##1'.\\%
4424       That's usually fine, because many languages\\%
4425       require no specific features, but if the output is\\%
4426       not as expected, consider selecting another font.}
4427     \expandafter
4428     \def\csname msg~text~>~fontspec/no-script\endcsname##1##2##3##4{%
4429       Font '\l_fontspec_fontname_tl' is using the\\%
4430       default features for script '##2'.\\%
4431       That's not always wrong, but if the output is\\%
4432       not as expected, consider selecting another font.}}
4433   \ExplSyntaxOff
4434 \fi
4435 \@onlypreamble\babelfont
4436 \newcommand\babelfont[2][]{%  1=langs/scripts 2=fam
4437   \bbl@foreach{#1}{%
4438     \expandafter\ifx\csname date##1\endcsname\relax
4439       \IfFileExists{babel-##1.tex}%
4440         {\babelprovide{##1}}%
4441         {}%
4442     \fi}%
4443   \edef\bbl@tempa{#1}%
4444   \def\bbl@tempb{#2}%  Used by \bbl@bblfont
4445   \ifx\fontspec\@undefined
4446     \bbl@loadfontspec
4447   \fi
4448   \EnableBabelHook{babel-fontspec}% Just calls \bbl@switchfont
4449   \bbl@bblfont}
4450 \newcommand\bbl@bblfont[2][]{% 1=features 2=fontname, @font=rm|sf|tt
4451   \bbl@ifunset{\bbl@tempb family}%
4452     {\bbl@providefam{\bbl@tempb}}%
4453     {}%
4454   % For the default font, just in case:
4455   \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
4456   \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
4457     {\bbl@csarg\edef{\bbl@tempb dflt@}{<>{#1}{#2}}% save bbl@rmdflt@
4458       \bbl@exp{%
4459         \let\<bbl@\bbl@tempb dflt@\languagename>\<bbl@\bbl@tempb dflt@>%
4460         \\\bbl@font@set\<bbl@\bbl@tempb dflt@\languagename>%
```

153

```
4461                    \<\bbl@tempb default>\<\bbl@tempb family>}}%
4462     {\bbl@foreach\bbl@tempa{%  ie bbl@rmdflt@lang / *scrt
4463         \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:

```
4464 \def\bbl@providefam#1{%
4465   \bbl@exp{%
4466     \\\newcommand\<#1default>{}% Just define it
4467     \\\bbl@add@list\\\bbl@font@fams{#1}%
4468     \\\DeclareRobustCommand\<#1family>{%
4469       \\\not@math@alphabet\<#1family>\relax
4470       % \\\prepare@family@series@update{#1}\<#1default>% TODO. Fails
4471       \\\fontfamily\<#1default>%
4472       \<\ifx>\\\UseHooks\\\@undefined\<\else>\\\UseHook{#1family}\<\fi>%
4473       \\\selectfont}%
4474     \\\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.

```
4475 \def\bbl@nostdfont#1{%
4476   \bbl@ifunset{bbl@WFF@\f@family}%
4477     {\bbl@csarg\gdef{WFF@\f@family}{}%  Flag, to avoid dupl warns
4478     \bbl@infowarn{The current font is not a babel standard family:\\%
4479       #1%
4480       \fontname\font\\%
4481       There is nothing intrinsically wrong with this warning, and\\%
4482       you can ignore it altogether if you do not need these\\%
4483       families. But if they are used in the document, you should be\\%
4484       aware 'babel' will no set Script and Language for them, so\\%
4485       you may consider defining a new family with \string\babelfont.\\%
4486       See the manual for further details about \string\babelfont.\\%
4487       Reported}}%
4488     {}}%
4489 \gdef\bbl@switchfont{%
4490   \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
4491   \bbl@exp{%  eg Arabic -> arabic
4492     \lowercase{\edef\\\bbl@tempa{\bbl@cl{sname}}}}%
4493   \bbl@foreach\bbl@font@fams{%
4494     \bbl@ifunset{bbl@##1dflt@\languagename}%     (1) language?
4495       {\bbl@ifunset{bbl@##1dflt@*\bbl@tempa}%   (2) from script?
4496         {\bbl@ifunset{bbl@##1dflt@}%            2=F - (3) from generic?
4497           {}%                                   123=F - nothing!
4498           {\bbl@exp{%                           3=T - from generic
4499             \global\let\<bbl@##1dflt@\languagename>%
4500                        \<bbl@##1dflt@>}}}%
4501         {\bbl@exp{%                             2=T - from script
4502           \global\let\<bbl@##1dflt@\languagename>%
4503                      \<bbl@##1dflt@*\bbl@tempa>}}}%
4504       {}}%                                      1=T - language, already defined
4505   \def\bbl@tempa{\bbl@nostdfont{}}%
4506   \bbl@foreach\bbl@font@fams{%      don't gather with prev for
4507     \bbl@ifunset{bbl@##1dflt@\languagename}%
4508       {\bbl@cs{famrst@##1}%
4509        \global\bbl@csarg\let{famrst@##1}\relax}%
4510       {\bbl@exp{% order is relevant. TODO: but sometimes wrong!
4511         \\\bbl@add\\\originalTeX{%
4512           \\\bbl@font@rst{\bbl@cl{##1dflt}}%
4513                          \<##1default>\<##1family>{##1}}%
4514         \\\bbl@font@set\<bbl@##1dflt@\languagename>% the main part!
4515                        \<##1default>\<##1family>}}}%
4516   \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.

```
4517 \ifx\f@family\@undefined\else    % if latex
4518   \ifcase\bbl@engine             % if pdftex
4519     \let\bbl@ckeckstdfonts\relax
4520   \else
4521     \def\bbl@ckeckstdfonts{%
4522       \begingroup
4523         \global\let\bbl@ckeckstdfonts\relax
4524         \let\bbl@tempa\@empty
4525         \bbl@foreach\bbl@font@fams{%
4526           \bbl@ifunset{bbl@##1dflt@}%
4527             {\@nameuse{##1family}%
4528              \bbl@csarg\gdef{WFF@\f@family}{}% Flag
4529              \bbl@exp{\\\bbl@add\\\bbl@tempa{* \<##1family>= \f@family\\\\%
4530                  \space\space\fontname\font\\\\}}%
4531              \bbl@csarg\xdef{##1dflt@}{\f@family}%
4532              \expandafter\xdef\csname ##1default\endcsname{\f@family}}%
4533             {}}%
4534         \ifx\bbl@tempa\@empty\else
4535           \bbl@infowarn{The following font families will use the default\\%
4536             settings for all or some languages:\\%
4537             \bbl@tempa
4538             There is nothing intrinsically wrong with it, but\\%
4539             'babel' will no set Script and Language, which could\\%
4540              be relevant in some languages. If your document uses\\%
4541              these families, consider redefining them with \string\babelfont.\\%
4542             Reported}%
4543         \fi
4544       \endgroup}
4545   \fi
4546 \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.

```
4547 \def\bbl@font@set#1#2#3{% eg \bbl@rmdflt@lang \rmdefault \rmfamily
4548   \bbl@xin@{<>}{#1}%
4549   \ifin@
4550     \bbl@exp{\\\bbl@fontspec@set\\#1\expandafter\@gobbletwo#1\\#3}%
4551   \fi
4552   \bbl@exp{%              'Unprotected' macros return prev values
4553     \def\\#2{#1}%          eg, \rmdefault{\bbl@rmdflt@lang}
4554     \\\bbl@ifsamestring{#2}{\f@family}%
4555       {\\#3%
4556        \\\bbl@ifsamestring{\f@series}{\bfdefault}{\\\bfseries}{}%
4557        \let\\\bbl@tempa\relax}%
4558       {}}}
4559 %     TODO - next should be global?, but even local does its job. I'm
4560 %     still not sure -- must investigate:
4561 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
4562   \let\bbl@tempe\bbl@mapselect
4563   \let\bbl@mapselect\relax
4564   \let\bbl@temp@fam#4%        eg, '\rmfamily', to be restored below
4565   \let#4\@empty       %       Make sure \renewfontfamily is valid
4566   \bbl@exp{%
4567     \let\\\bbl@temp@pfam\<\bbl@stripslash#4\space>% eg, '\rmfamily '
4568     \<keys_if_exist:nnF>{fontspec-opentype}{Script/\bbl@cl{sname}}%
4569       {\\\newfontscript{\bbl@cl{sname}}{\bbl@cl{sotf}}}%
4570     \<keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@cl{lname}}%
4571       {\\\newfontlanguage{\bbl@cl{lname}}{\bbl@cl{lotf}}}%
4572     \\\renewfontfamily\\#4%
4573       [\bbl@cl{lsys},#2]}{#3}% ie \bbl@exp{..}{#3}
4574   \begingroup
```

```
4575        #4%
4576        \xdef#1{\f@family}%      eg, \bbl@rmdflt@lang{FreeSerif(0)}
4577    \endgroup
4578    \let#4\bbl@temp@fam
4579    \bbl@exp{\let\<\bbl@stripslash#4\space>}\bbl@temp@pfam
4580    \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.

```
4581 \def\bbl@font@rst#1#2#3#4{%
4582    \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.

```
4583 \def\bbl@font@fams{rm,sf,tt}
```

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

```
4584 \newcommand\babelFSstore[2][]{%
4585    \bbl@ifblank{#1}%
4586        {\bbl@csarg\def{sname@#2}{Latin}}%
4587        {\bbl@csarg\def{sname@#2}{#1}}%
4588    \bbl@provide@dirs{#2}%
4589    \bbl@csarg\ifnum{wdir@#2}>\z@
4590        \let\bbl@beforeforeign\leavevmode
4591        \EnableBabelHook{babel-bidi}%
4592    \fi
4593    \bbl@foreach{#2}{%
4594        \bbl@FSstore{##1}{rm}\rmdefault\bbl@save@rmdefault
4595        \bbl@FSstore{##1}{sf}\sfdefault\bbl@save@sfdefault
4596        \bbl@FSstore{##1}{tt}\ttdefault\bbl@save@ttdefault}}
4597 \def\bbl@FSstore#1#2#3#4{%
4598    \bbl@csarg\edef{#2default#1}{#3}%
4599    \expandafter\addto\csname extras#1\endcsname{%
4600        \let#4#3%
4601        \ifx#3\f@family
4602            \edef#3{\csname bbl@#2default#1\endcsname}%
4603            \fontfamily{#3}\selectfont
4604        \else
4605            \edef#3{\csname bbl@#2default#1\endcsname}%
4606        \fi}%
4607    \expandafter\addto\csname noextras#1\endcsname{%
4608        \ifx#3\f@family
4609            \fontfamily{#4}\selectfont
4610        \fi
4611        \let#3#4}}
4612 \let\bbl@langfeatures\@empty
4613 \def\babelFSfeatures{% make sure \fontspec is redefined once
4614    \let\bbl@ori@fontspec\fontspec
4615    \renewcommand\fontspec[1][]{%
4616        \bbl@ori@fontspec[\bbl@langfeatures##1]}%
4617    \let\babelFSfeatures\bbl@FSfeatures
4618    \babelFSfeatures}
4619 \def\bbl@FSfeatures#1#2{%
4620    \expandafter\addto\csname extras#1\endcsname{%
4621        \babel@save\bbl@langfeatures
4622        \edef\bbl@langfeatures{#2,}}}
4623 ⟨⟨/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.

```
4624 ⟨⟨*Footnote changes⟩⟩ ≡
4625 \bbl@trace{Bidi footnotes}
4626 \ifnum\bbl@bidimode>\z@
4627   \def\bbl@footnote#1#2#3{%
4628     \@ifnextchar[%
4629       {\bbl@footnote@o{#1}{#2}{#3}}%
4630       {\bbl@footnote@x{#1}{#2}{#3}}}
4631   \long\def\bbl@footnote@x#1#2#3#4{%
4632     \bgroup
4633       \select@language@x{\bbl@main@language}%
4634       \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
4635     \egroup}
4636   \long\def\bbl@footnote@o#1#2#3[#4]#5{%
4637     \bgroup
4638       \select@language@x{\bbl@main@language}%
4639       \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
4640     \egroup}
4641   \def\bbl@footnotetext#1#2#3{%
4642     \@ifnextchar[%
4643       {\bbl@footnotetext@o{#1}{#2}{#3}}%
4644       {\bbl@footnotetext@x{#1}{#2}{#3}}}
4645   \long\def\bbl@footnotetext@x#1#2#3#4{%
4646     \bgroup
4647       \select@language@x{\bbl@main@language}%
4648       \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
4649     \egroup}
4650   \long\def\bbl@footnotetext@o#1#2#3[#4]#5{%
4651     \bgroup
4652       \select@language@x{\bbl@main@language}%
4653       \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
4654     \egroup}
4655   \def\BabelFootnote#1#2#3#4{%
4656     \ifx\bbl@fn@footnote\@undefined
4657       \let\bbl@fn@footnote\footnote
4658     \fi
4659     \ifx\bbl@fn@footnotetext\@undefined
4660       \let\bbl@fn@footnotetext\footnotetext
4661     \fi
4662     \bbl@ifblank{#2}%
4663       {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
4664        \@namedef{\bbl@stripslash#1text}%
4665          {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
4666       {\def#1{\bbl@exp{\\\bbl@footnote{\\\foreignlanguage{#2}}}{#3}{#4}}%
4667        \@namedef{\bbl@stripslash#1text}%
4668          {\bbl@exp{\\\bbl@footnotetext{\\\foreignlanguage{#2}}}{#3}{#4}}}}
4669 \fi
4670 ⟨⟨/Footnote changes⟩⟩
```

Now, the code.

```
4671 ⟨*xetex⟩
4672 \def\BabelStringsDefault{unicode}
4673 \let\xebbl@stop\relax
4674 \AddBabelHook{xetex}{encodedcommands}{%
4675   \def\bbl@tempa{#1}%
4676   \ifx\bbl@tempa\@empty
4677     \XeTeXinputencoding"bytes"%
4678   \else
```

```
4679      \XeTeXinputencoding"#1"%
4680   \fi
4681   \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4682 \AddBabelHook{xetex}{stopcommands}{%
4683   \xebbl@stop
4684   \let\xebbl@stop\relax}
4685 \def\bbl@intraspace#1 #2 #3\@@{%
4686   \bbl@csarg\gdef{xeisp@\languagename}%
4687      {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4688 \def\bbl@intrapenalty#1\@@{%
4689   \bbl@csarg\gdef{xeipn@\languagename}%
4690      {\XeTeXlinebreakpenalty #1\relax}}
4691 \def\bbl@provide@intraspace{%
4692   \bbl@xin@{/s}{/\bbl@cl{lnbrk}}%
4693   \ifin@\else\bbl@xin@{/c}{/\bbl@cl{lnbrk}}\fi
4694   \ifin@
4695     \bbl@ifunset{bbl@intsp@\languagename}{}%
4696        {\expandafter\ifx\csname bbl@intsp@\languagename\endcsname\@empty\else
4697          \ifx\bbl@KVP@intraspace\@nil
4698             \bbl@exp{%
4699                \\\bbl@intraspace\bbl@cl{intsp}\\\@@}%
4700          \fi
4701          \ifx\bbl@KVP@intrapenalty\@nil
4702             \bbl@intrapenalty0\@@
4703          \fi
4704       \fi
4705       \ifx\bbl@KVP@intraspace\@nil\else % We may override the ini
4706          \expandafter\bbl@intraspace\bbl@KVP@intraspace\@@
4707       \fi
4708       \ifx\bbl@KVP@intrapenalty\@nil\else
4709          \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
4710       \fi
4711       \bbl@exp{%
4712          % TODO. Execute only once (but redundant):
4713          \\\bbl@add\<extras\languagename>{%
4714             \XeTeXlinebreaklocale "\bbl@cl{tbcp}"%
4715             \<bbl@xeisp@\languagename>%
4716             \<bbl@xeipn@\languagename>}%
4717          \\\bbl@toglobal\<extras\languagename>%
4718          \\\bbl@add\<noextras\languagename>{%
4719             \XeTeXlinebreaklocale "en"}%
4720          \\\bbl@toglobal\<noextras\languagename>}%
4721       \ifx\bbl@ispacesize\@undefined
4722          \gdef\bbl@ispacesize{\bbl@cl{xeisp}}%
4723          \ifx\AtBeginDocument\@notprerr
4724             \expandafter\@secondoftwo  % to execute right now
4725          \fi
4726          \AtBeginDocument{\bbl@patchfont{\bbl@ispacesize}}%
4727       \fi}%
4728   \fi}
4729 \ifx\DisableBabelHook\@undefined\endinput\fi
4730 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4731 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
4732 \DisableBabelHook{babel-fontspec}
4733 ⟨⟨Font selection⟩⟩
4734 \input txtbabel.def
4735 ⟨/xetex⟩
```

12.2 Layout

In progress.
Note elements like headlines and margins can be modified easily with packages like fancyhdr, typearea or titleps, and geometry.

\bbl@startskip and \bbl@endskip are available to package authors. Thanks to the TEX expansion mechanism the following constructs are valid: \adim\bbl@startskip, \advance\bbl@startskip\adim, \bbl@startskip\adim.

Consider txtbabel as a shorthand for *tex–xet babel*, which is the bidi model in both pdftex and xetex.

```
4736 ⟨*texxet⟩
4737 \providecommand\bbl@provide@intraspace{}
4738 \bbl@trace{Redefinitions for bidi layout}
4739 \def\bbl@sspre@caption{%
4740   \bbl@exp{\everyhbox{\\\bbl@textdir\bbl@cs{wdir@\bbl@main@language}}}}
4741 \ifx\bbl@opt@layout\@nnil\endinput\fi  % No layout
4742 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
4743 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
4744 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
4745   \def\@hangfrom#1{%
4746     \setbox\@tempboxa\hbox{{#1}}%
4747     \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
4748     \noindent\box\@tempboxa}
4749   \def\raggedright{%
4750     \let\\\@centercr
4751     \bbl@startskip\z@skip
4752     \@rightskip\@flushglue
4753     \bbl@endskip\@rightskip
4754     \parindent\z@
4755     \parfillskip\bbl@startskip}
4756   \def\raggedleft{%
4757     \let\\\@centercr
4758     \bbl@startskip\@flushglue
4759     \bbl@endskip\z@skip
4760     \parindent\z@
4761     \parfillskip\bbl@endskip}
4762 \fi
4763 \IfBabelLayout{lists}
4764   {\bbl@sreplace\list
4765     {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
4766   \def\bbl@listleftmargin{%
4767     \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
4768   \ifcase\bbl@engine
4769     \def\labelenumii{)\theenumii(}% pdftex doesn't reverse ()
4770     \def\p@enumiii{\p@enumii)\theenumii(}%
4771   \fi
4772   \bbl@sreplace\@verbatim
4773     {\leftskip\@totalleftmargin}%
4774     {\bbl@startskip\textwidth
4775      \advance\bbl@startskip-\linewidth}%
4776   \bbl@sreplace\@verbatim
4777     {\rightskip\z@skip}%
4778     {\bbl@endskip\z@skip}}%
4779   {}
4780 \IfBabelLayout{contents}
4781   {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
4782    \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
4783   {}
4784 \IfBabelLayout{columns}
4785   {\bbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbl@outputhbox}%
4786   \def\bbl@outputhbox#1{%
4787     \hb@xt@\textwidth{%
4788       \hskip\columnwidth
4789       \hfil
4790       {\normalcolor\vrule \@width\columnseprule}%
4791       \hfil
4792       \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
4793       \hskip-\textwidth
4794       \hb@xt@\columnwidth{\box\@outputbox \hss}%
```

```
4795        \hskip\columnsep
4796        \hskip\columnwidth}}}%
4797    {}
```
4798 ⟨⟨*Footnote changes*⟩⟩
```
4799 \IfBabelLayout{footnotes}%
4800    {\BabelFootnote\footnote\languagename{}{}%
4801     \BabelFootnote\localfootnote\languagename{}{}%
4802     \BabelFootnote\mainfootnote{}{}{}}
4803    {}
```

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.

```
4804 \IfBabelLayout{counters}%
4805    {\let\bbl@latinarabic=\@arabic
4806     \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
4807     \let\bbl@asciiroman=\@roman
4808     \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciiroman#1}}}%
4809     \let\bbl@asciiRoman=\@Roman
4810     \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}}{}
```
4811 ⟨/texxet⟩

12.3 LuaTeX

The loader for luatex is based solely on language.dat, which is read on the fly. The code shouldn't be executed when the format is build, so we check if \AddBabelHook is defined. Then comes a modified version of the loader in hyphen.cfg (without the hyphenmins stuff, which is under the direct control of babel).

The names \l@<language> are defined and take some value from the beginning because all ldf files assume this for the corresponding language to be considered valid, but patterns are not loaded (except the first one). This is done later, when the language is first selected (which usually means when the ldf finishes). If a language has been loaded, \bbl@hyphendata@<num> exists (with the names of the files read).

The default setup preloads the first language into the format. This is intended mainly for 'english', so that it's available without further intervention from the user. To avoid duplicating it, the following rule applies: if the "0th" language and the first language in language.dat have the same name then just ignore the latter. If there are new synonymous, the are added, but note if the language patterns have not been preloaded they won't at run time.

Other preloaded languages could be read twice, if they have been preloaded into the format. This is not optimal, but it shouldn't happen very often – with luatex patterns are best loaded when the document is typeset, and the "0th" language is preloaded just for backwards compatibility.

As of 1.1b, lua(e)tex is taken into account. Formerly, loading of patterns on the fly didn't work in this format, but with the new loader it does. Unfortunately, the format is not based on babel, and data could be duplicated, because languages are reassigned above those in the format (nothing serious, anyway). Note even with this format language.dat is used (under the principle of a single source), instead of language.def.

Of course, there is room for improvements, like tools to read and reassign languages, which would require modifying the language list, and better error handling.

We need catcode tables, but no format (targeted by babel) provide a command to allocate them (although there are packages like ctablestack). FIX - This isn't true anymore. For the moment, a dangerous approach is used - just allocate a high random number and cross the fingers. To complicate things, etex.sty changes the way languages are allocated.

This files is read at three places: (1) when plain.def, babel.sty starts, to read the list of available languages from language.dat (for the base option); (2) at hyphen.cfg, to modify some macros; (3) in the middle of plain.def and babel.sty, by babel.def, with the commands and other definitions for luatex (eg, \babelpatterns).

```
4812 ⟨*luatex⟩
4813 \ifx\AddBabelHook\@undefined % When plain.def, babel.sty starts
4814 \bbl@trace{Read language.dat}
4815 \ifx\bbl@readstream\@undefined
4816   \csname newread\endcsname\bbl@readstream
4817 \fi
4818 \begingroup
4819   \toks@{}
```

```
4820    \count@\z@ % 0=start, 1=0th, 2=normal
4821    \def\bbl@process@line#1#2 #3 #4 {%
4822      \ifx=#1%
4823        \bbl@process@synonym{#2}%
4824      \else
4825        \bbl@process@language{#1#2}{#3}{#4}%
4826      \fi
4827      \ignorespaces}
4828    \def\bbl@manylang{%
4829      \ifnum\bbl@last>\@ne
4830        \bbl@info{Non-standard hyphenation setup}%
4831      \fi
4832      \let\bbl@manylang\relax}
4833    \def\bbl@process@language#1#2#3{%
4834      \ifcase\count@
4835        \@ifundefined{zth@#1}{\count@\tw@}{\count@\@ne}%
4836      \or
4837        \count@\tw@
4838      \fi
4839      \ifnum\count@=\tw@
4840        \expandafter\addlanguage\csname l@#1\endcsname
4841        \language\allocationnumber
4842        \chardef\bbl@last\allocationnumber
4843        \bbl@manylang
4844        \let\bbl@elt\relax
4845        \xdef\bbl@languages{%
4846          \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
4847      \fi
4848      \the\toks@
4849      \toks@{}}
4850    \def\bbl@process@synonym@aux#1#2{%
4851      \global\expandafter\chardef\csname l@#1\endcsname#2\relax
4852      \let\bbl@elt\relax
4853      \xdef\bbl@languages{%
4854        \bbl@languages\bbl@elt{#1}{#2}{}{}}%
4855    \def\bbl@process@synonym#1{%
4856      \ifcase\count@
4857        \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
4858      \or
4859        \@ifundefined{zth@#1}{\bbl@process@synonym@aux{#1}{0}}{}%
4860      \else
4861        \bbl@process@synonym@aux{#1}{\the\bbl@last}%
4862      \fi}
4863    \ifx\bbl@languages\@undefined % Just a (sensible?) guess
4864      \chardef\l@english\z@
4865      \chardef\l@USenglish\z@
4866      \chardef\bbl@last\z@
4867      \global\@namedef{bbl@hyphendata@0}{{hyphen.tex}{}}
4868      \gdef\bbl@languages{%
4869        \bbl@elt{english}{0}{hyphen.tex}{}%
4870        \bbl@elt{USenglish}{0}{}{}}
4871    \else
4872      \global\let\bbl@languages@format\bbl@languages
4873      \def\bbl@elt#1#2#3#4{% Remove all except language 0
4874        \ifnum#2>\z@\else
4875          \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
4876        \fi}%
4877      \xdef\bbl@languages{\bbl@languages}%
4878    \fi
4879    \def\bbl@elt#1#2#3#4{\@namedef{zth@#1}{}} % Define flags
4880    \bbl@languages
4881    \openin\bbl@readstream=language.dat
4882    \ifeof\bbl@readstream
```

161

```
4883      \bbl@warning{I couldn't find language.dat. No additional\\%
4884                  patterns loaded. Reported}%
4885    \else
4886      \loop
4887        \endlinechar\m@ne
4888        \read\bbl@readstream to \bbl@line
4889        \endlinechar`\^^M
4890        \if T\ifeof\bbl@readstream F\fi T\relax
4891          \ifx\bbl@line\@empty\else
4892            \edef\bbl@line{\bbl@line\space\space\space}%
4893            \expandafter\bbl@process@line\bbl@line\relax
4894          \fi
4895      \repeat
4896    \fi
4897 \endgroup
4898 \bbl@trace{Macros for reading patterns files}
4899 \def\bbl@get@enc#1:#2:#3\@@@{\def\bbl@hyph@enc{#2}}
4900 \ifx\babelcatcodetablenum\@undefined
4901   \ifx\newcatcodetable\@undefined
4902     \def\babelcatcodetablenum{5211}
4903     \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4904   \else
4905     \newcatcodetable\babelcatcodetablenum
4906     \newcatcodetable\bbl@pattcodes
4907   \fi
4908 \else
4909   \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4910 \fi
4911 \def\bbl@luapatterns#1#2{%
4912   \bbl@get@enc#1::\@@@
4913   \setbox\z@\hbox\bgroup
4914     \begingroup
4915       \savecatcodetable\babelcatcodetablenum\relax
4916       \initcatcodetable\bbl@pattcodes\relax
4917       \catcodetable\bbl@pattcodes\relax
4918         \catcode`\#=6  \catcode`\$=3 \catcode`\&=4 \catcode`\^=7
4919         \catcode`\_=8  \catcode`\{=1 \catcode`\}=2 \catcode`\~=13
4920         \catcode`\@=11 \catcode`\^^I=10 \catcode`\^^J=12
4921         \catcode`\<=12 \catcode`\>=12 \catcode`\*=12 \catcode`\.=12
4922         \catcode`\-=12 \catcode`\/=12 \catcode`\[=12 \catcode`\]=12
4923         \catcode`\`=12 \catcode`\'=12 \catcode`\"=12
4924         \input #1\relax
4925       \catcodetable\babelcatcodetablenum\relax
4926     \endgroup
4927     \def\bbl@tempa{#2}%
4928     \ifx\bbl@tempa\@empty\else
4929       \input #2\relax
4930     \fi
4931   \egroup}%
4932 \def\bbl@patterns@lua#1{%
4933   \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
4934     \csname l@#1\endcsname
4935     \edef\bbl@tempa{#1}%
4936   \else
4937     \csname l@#1:\f@encoding\endcsname
4938     \edef\bbl@tempa{#1:\f@encoding}%
4939   \fi\relax
4940   \@namedef{lu@texhyphen@loaded@\the\language}{}% Temp
4941   \@ifundefined{bbl@hyphendata@\the\language}%
4942     {\def\bbl@elt##1##2##3##4{%
4943       \ifnum##2=\csname l@\bbl@tempa\endcsname % #2=spanish, dutch:OT1...
4944         \def\bbl@tempb{##3}%
4945         \ifx\bbl@tempb\@empty\else % if not a synonymous
```

162

```
4946        \def\bbl@tempc{{##3}{##4}}%
4947          \fi
4948          \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
4949        \fi}%
4950      \bbl@languages
4951      \@ifundefined{bbl@hyphendata@\the\language}%
4952        {\bbl@info{No hyphenation patterns were set for\\%
4953                  language '\bbl@tempa'. Reported}}%
4954        {\expandafter\expandafter\expandafter\bbl@luapatterns
4955          \csname bbl@hyphendata@\the\language\endcsname}}{}}
4956 \endinput\fi
4957   % Here ends \ifx\AddBabelHook\@undefined
4958   % A few lines are only read by hyphen.cfg
4959 \ifx\DisableBabelHook\@undefined
4960   \AddBabelHook{luatex}{everylanguage}{%
4961     \def\process@language##1##2##3{%
4962       \def\process@line####1####2 ####3 ####4 {}}}
4963   \AddBabelHook{luatex}{loadpatterns}{%
4964     \input #1\relax
4965     \expandafter\gdef\csname bbl@hyphendata@\the\language\endcsname
4966       {{#1}{}}}
4967   \AddBabelHook{luatex}{loadexceptions}{%
4968     \input #1\relax
4969     \def\bbl@tempb##1##2{{##1}{#1}}%
4970     \expandafter\xdef\csname bbl@hyphendata@\the\language\endcsname
4971       {\expandafter\expandafter\expandafter\bbl@tempb
4972        \csname bbl@hyphendata@\the\language\endcsname}}
4973 \endinput\fi
4974   % Here stops reading code for hyphen.cfg
4975   % The following is read the 2nd time it's loaded
4976 \begingroup  % TODO - to a lua file
4977 \catcode`\%=12
4978 \catcode`\'=12
4979 \catcode`\"=12
4980 \catcode`\:=12
4981 \directlua{
4982   Babel = Babel or {}
4983   function Babel.bytes(line)
4984     return line:gsub("(.)",
4985       function (chr) return unicode.utf8.char(string.byte(chr)) end)
4986   end
4987   function Babel.begin_process_input()
4988     if luatexbase and luatexbase.add_to_callback then
4989       luatexbase.add_to_callback('process_input_buffer',
4990                                  Babel.bytes,'Babel.bytes')
4991     else
4992       Babel.callback = callback.find('process_input_buffer')
4993       callback.register('process_input_buffer',Babel.bytes)
4994     end
4995   end
4996   function Babel.end_process_input ()
4997     if luatexbase and luatexbase.remove_from_callback then
4998       luatexbase.remove_from_callback('process_input_buffer','Babel.bytes')
4999     else
5000       callback.register('process_input_buffer',Babel.callback)
5001     end
5002   end
5003   function Babel.addpatterns(pp, lg)
5004     local lg = lang.new(lg)
5005     local pats = lang.patterns(lg) or ''
5006     lang.clear_patterns(lg)
5007     for p in pp:gmatch('[^%s]+') do
5008       ss = ''
```

```
5009        for i in string.utfcharacters(p:gsub('%d', '')) do
5010          ss = ss .. '%d?' .. i
5011        end
5012        ss = ss:gsub('^%%d%?%.', '%%.') .. '%d?'
5013        ss = ss:gsub('%.%%d%?$', '%%.')
5014        pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
5015        if n == 0 then
5016          tex.sprint(
5017            [[\string\csname\space bbl@info\endcsname{New pattern: ]]
5018            .. p .. [[}]])
5019          pats = pats .. ' ' .. p
5020        else
5021          tex.sprint(
5022            [[\string\csname\space bbl@info\endcsname{Renew pattern: ]]
5023            .. p .. [[}]])
5024        end
5025      end
5026      lang.patterns(lg, pats)
5027    end
5028    function Babel.hlist_has_bidi(head)
5029      local has_bidi = false
5030      for item in node.traverse(head) do
5031        if item.id == node.id'glyph' then
5032          local itemchar = item.char
5033          local chardata = Babel.characters[itemchar]
5034          local dir = chardata and chardata.d or nil
5035          if not dir then
5036            for nn, et in ipairs(Babel.ranges) do
5037              if itemchar < et[1] then
5038                break
5039              elseif itemchar <= et[2] then
5040                dir = et[3]
5041                break
5042              end
5043            end
5044          end
5045          if dir and (dir == 'al' or dir == 'r') then
5046            has_bidi = true
5047          end
5048        end
5049      end
5050      return has_bidi
5051    end
5052    function Babel.set_chranges_b (script, chrng)
5053      if chrng == '' then return end
5054      texio.write('Replacing ' .. script .. ' script ranges')
5055      Babel.script_blocks[script] = {}
5056      for s, e in string.gmatch(chrng..' ', '(.-)%.%.(.-)%s') do
5057        table.insert(
5058          Babel.script_blocks[script], {tonumber(s,16), tonumber(e,16)})
5059      end
5060    end
5061 }
5062 \endgroup
5063 \ifx\newattribute\@undefined\else
5064   \newattribute\bbl@attr@locale
5065   \directlua{ Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale' }
5066   \AddBabelHook{luatex}{beforeextras}{%
5067     \setattribute\bbl@attr@locale\localeid}
5068 \fi
5069 \def\BabelStringsDefault{unicode}
5070 \let\luabbl@stop\relax
5071 \AddBabelHook{luatex}{encodedcommands}{%
```

```
5072    \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
5073    \ifx\bbl@tempa\bbl@tempb\else
5074      \directlua{Babel.begin_process_input()}%
5075      \def\luabbl@stop{%
5076        \directlua{Babel.end_process_input()}}%
5077    \fi}%
5078  \AddBabelHook{luatex}{stopcommands}{%
5079    \luabbl@stop
5080    \let\luabbl@stop\relax}
5081  \AddBabelHook{luatex}{patterns}{%
5082    \@ifundefined{bbl@hyphendata@\the\language}%
5083      {\def\bbl@elt##1##2##3##4{%
5084         \ifnum##2=\csname l@##2\endcsname % #2=spanish, dutch:OT1...
5085           \def\bbl@tempb{##3}%
5086           \ifx\bbl@tempb\@empty\else % if not a synonymous
5087             \def\bbl@tempc{{##3}{##4}}%
5088           \fi
5089           \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5090         \fi}%
5091       \bbl@languages
5092       \@ifundefined{bbl@hyphendata@\the\language}%
5093         {\bbl@info{No hyphenation patterns were set for\\%
5094                    language '#2'. Reported}}%
5095         {\expandafter\expandafter\expandafter\bbl@luapatterns
5096           \csname bbl@hyphendata@\the\language\endcsname}}{}%
5097    \@ifundefined{bbl@patterns@}{}{%
5098      \begingroup
5099        \bbl@xin@{,\number\language,}{,\bbl@pttnlist}%
5100        \ifin@\else
5101          \ifx\bbl@patterns@\@empty\else
5102            \directlua{ Babel.addpatterns(
5103              [[\bbl@patterns@]], \number\language) }%
5104          \fi
5105          \@ifundefined{bbl@patterns@#1}%
5106            \@empty
5107            {\directlua{ Babel.addpatterns(
5108                [[\space\csname bbl@patterns@#1\endcsname]],
5109                \number\language) }}%
5110          \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
5111        \fi
5112      \endgroup}%
5113    \bbl@exp{%
5114      \bbl@ifunset{bbl@prehc@\languagename}{}%
5115        {\\\bbl@ifblank{\bbl@cs{prehc@\languagename}}{}%
5116          {\prehyphenchar=\bbl@cl{prehc}\relax}}}}
```

\babelpatterns   This macro adds patterns. Two macros are used to store them: \bbl@patterns@ for the global ones
and \bbl@patterns@<lang> for language ones. We make sure there is a space between words when
multiple commands are used.

```
5117  \@onlypreamble\babelpatterns
5118  \AtEndOfPackage{%
5119    \newcommand\babelpatterns[2][\@empty]{%
5120      \ifx\bbl@patterns@\relax
5121        \let\bbl@patterns@\@empty
5122      \fi
5123      \ifx\bbl@pttnlist\@empty\else
5124        \bbl@warning{%
5125          You must not intermingle \string\selectlanguage\space and\\%
5126          \string\babelpatterns\space or some patterns will not\\%
5127          be taken into account. Reported}%
5128      \fi
5129      \ifx\@empty#1%
5130        \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
```

```
5131      \else
5132        \edef\bbl@tempb{\zap@space#1 \@empty}%
5133        \bbl@for\bbl@tempa\bbl@tempb{%
5134          \bbl@fixname\bbl@tempa
5135          \bbl@iflanguage\bbl@tempa{%
5136            \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
5137              \@ifundefined{bbl@patterns@\bbl@tempa}%
5138                \@empty
5139                {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
5140            #2}}}%
5141      \fi}}
```

12.4 Southeast Asian scripts

First, some general code for line breaking, used by \babelposthyphenation.
Replace regular (ie, implicit) discretionaries by spaceskips, based on the previous glyph (which I think makes sense, because the hyphen and the previous char go always together). Other discretionaries are not touched. See Unicode UAX 14.

```
5142 % TODO - to a lua file
5143 \directlua{
5144   Babel = Babel or {}
5145   Babel.linebreaking = Babel.linebreaking or {}
5146   Babel.linebreaking.before = {}
5147   Babel.linebreaking.after = {}
5148   Babel.locale = {} % Free to use, indexed by \localeid
5149   function Babel.linebreaking.add_before(func)
5150     tex.print([[\noexpand\csname bbl@luahyphenate\endcsname]])
5151     table.insert(Babel.linebreaking.before, func)
5152   end
5153   function Babel.linebreaking.add_after(func)
5154     tex.print([[\noexpand\csname bbl@luahyphenate\endcsname]])
5155     table.insert(Babel.linebreaking.after, func)
5156   end
5157 }
5158 \def\bbl@intraspace#1 #2 #3\@@{%
5159   \directlua{
5160     Babel = Babel or {}
5161     Babel.intraspaces = Babel.intraspaces or {}
5162     Babel.intraspaces['\csname bbl@sbcp@\languagename\endcsname'] = %
5163       {b = #1, p = #2, m = #3}
5164     Babel.locale_props[\the\localeid].intraspace = %
5165       {b = #1, p = #2, m = #3}
5166   }}
5167 \def\bbl@intrapenalty#1\@@{%
5168   \directlua{
5169     Babel = Babel or {}
5170     Babel.intrapenalties = Babel.intrapenalties or {}
5171     Babel.intrapenalties['\csname bbl@sbcp@\languagename\endcsname'] = #1
5172     Babel.locale_props[\the\localeid].intrapenalty = #1
5173   }}
5174 \begingroup
5175 \catcode`\%=12
5176 \catcode`\^=14
5177 \catcode`\'=12
5178 \catcode`\~=12
5179 \gdef\bbl@seaintraspace{^
5180   \let\bbl@seaintraspace\relax
5181   \directlua{
5182     Babel = Babel or {}
5183     Babel.sea_enabled = true
5184     Babel.sea_ranges = Babel.sea_ranges or {}
5185     function Babel.set_chranges (script, chrng)
5186       local c = 0
```

```
5187       for s, e in string.gmatch(chrng..' ', '(.-)%.%.(.-)%s') do
5188         Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
5189         c = c + 1
5190       end
5191     end
5192     function Babel.sea_disc_to_space (head)
5193       local sea_ranges = Babel.sea_ranges
5194       local last_char = nil
5195       local quad = 655360        ^% 10 pt = 655360 = 10 * 65536
5196       for item in node.traverse(head) do
5197         local i = item.id
5198         if i == node.id'glyph' then
5199           last_char = item
5200         elseif i == 7 and item.subtype == 3 and last_char
5201             and last_char.char > 0x0C99 then
5202           quad = font.getfont(last_char.font).size
5203           for lg, rg in pairs(sea_ranges) do
5204             if last_char.char > rg[1] and last_char.char < rg[2] then
5205               lg = lg:sub(1, 4)  ^% Remove trailing number of, eg, Cyrl1
5206               local intraspace = Babel.intraspaces[lg]
5207               local intrapenalty = Babel.intrapenalties[lg]
5208               local n
5209               if intrapenalty ~= 0 then
5210                 n = node.new(14, 0)      ^% penalty
5211                 n.penalty = intrapenalty
5212                 node.insert_before(head, item, n)
5213               end
5214               n = node.new(12, 13)       ^% (glue, spaceskip)
5215               node.setglue(n, intraspace.b * quad,
5216                            intraspace.p * quad,
5217                            intraspace.m * quad)
5218               node.insert_before(head, item, n)
5219               node.remove(head, item)
5220             end
5221           end
5222         end
5223       end
5224     end
5225   }^^
5226   \bbl@luahyphenate}
```

12.5 CJK line breaking

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

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

```
5227 \catcode`\%=14
5228 \gdef\bbl@cjkintraspace{%
5229   \let\bbl@cjkintraspace\relax
5230   \directlua{
5231     Babel = Babel or {}
5232     require('babel-data-cjk.lua')
5233     Babel.cjk_enabled = true
5234     function Babel.cjk_linebreak(head)
5235       local GLYPH = node.id'glyph'
5236       local last_char = nil
5237       local quad = 655360        % 10 pt = 655360 = 10 * 65536
5238       local last_class = nil
5239       local last_lang = nil
5240
```

```
5241      for item in node.traverse(head) do
5242        if item.id == GLYPH then
5243
5244          local lang = item.lang
5245
5246          local LOCALE = node.get_attribute(item,
5247                Babel.attr_locale)
5248          local props = Babel.locale_props[LOCALE]
5249
5250          local class = Babel.cjk_class[item.char].c
5251
5252          if props.cjk_quotes and props.cjk_quotes[item.char] then
5253            class = props.cjk_quotes[item.char]
5254          end
5255
5256          if class == 'cp' then class = 'cl' end % )] as CL
5257          if class == 'id' then class = 'I' end
5258
5259          local br = 0
5260          if class and last_class and Babel.cjk_breaks[last_class][class] then
5261            br = Babel.cjk_breaks[last_class][class]
5262          end
5263
5264          if br == 1 and props.linebreak == 'c' and
5265              lang ~= \the\l@nohyphenation\space and
5266              last_lang ~= \the\l@nohyphenation then
5267            local intrapenalty = props.intrapenalty
5268            if intrapenalty ~= 0 then
5269              local n = node.new(14, 0)      % penalty
5270              n.penalty = intrapenalty
5271              node.insert_before(head, item, n)
5272            end
5273            local intraspace = props.intraspace
5274            local n = node.new(12, 13)       % (glue, spaceskip)
5275            node.setglue(n, intraspace.b * quad,
5276                            intraspace.p * quad,
5277                            intraspace.m * quad)
5278            node.insert_before(head, item, n)
5279          end
5280
5281          if font.getfont(item.font) then
5282            quad = font.getfont(item.font).size
5283          end
5284          last_class = class
5285          last_lang = lang
5286        else % if penalty, glue or anything else
5287          last_class = nil
5288        end
5289      end
5290      lang.hyphenate(head)
5291    end
5292  }%
5293  \bbl@luahyphenate}
5294 \gdef\bbl@luahyphenate{%
5295  \let\bbl@luahyphenate\relax
5296  \directlua{
5297    luatexbase.add_to_callback('hyphenate',
5298    function (head, tail)
5299      if Babel.linebreaking.before then
5300        for k, func in ipairs(Babel.linebreaking.before)  do
5301          func(head)
5302        end
5303      end
```

```
5304        if Babel.cjk_enabled then
5305          Babel.cjk_linebreak(head)
5306        end
5307        lang.hyphenate(head)
5308        if Babel.linebreaking.after then
5309          for k, func in ipairs(Babel.linebreaking.after)  do
5310            func(head)
5311          end
5312        end
5313        if Babel.sea_enabled then
5314          Babel.sea_disc_to_space(head)
5315        end
5316      end,
5317      'Babel.hyphenate')
5318    }
5319 }
5320 \endgroup
5321 \def\bbl@provide@intraspace{%
5322  \bbl@ifunset{bbl@intsp@\languagename}{}%
5323    {\expandafter\ifx\csname bbl@intsp@\languagename\endcsname\@empty\else
5324      \bbl@xin@{/c}{/\bbl@cl{lnbrk}}%
5325      \ifin@             % cjk
5326        \bbl@cjkintraspace
5327        \directlua{
5328            Babel = Babel or {}
5329            Babel.locale_props = Babel.locale_props or {}
5330            Babel.locale_props[\the\localeid].linebreak = 'c'
5331        }%
5332        \bbl@exp{\\\bbl@intraspace\bbl@cl{intsp}\\\@@}%
5333        \ifx\bbl@KVP@intrapenalty\@nil
5334          \bbl@intrapenalty0\@@
5335        \fi
5336      \else             % sea
5337        \bbl@seaintraspace
5338        \bbl@exp{\\\bbl@intraspace\bbl@cl{intsp}\\\@@}%
5339        \directlua{
5340          Babel = Babel or {}
5341          Babel.sea_ranges = Babel.sea_ranges or {}
5342          Babel.set_chranges('\bbl@cl{sbcp}',
5343                             '\bbl@cl{chrng}')
5344        }%
5345        \ifx\bbl@KVP@intrapenalty\@nil
5346          \bbl@intrapenalty0\@@
5347        \fi
5348      \fi
5349    \fi
5350    \ifx\bbl@KVP@intrapenalty\@nil\else
5351      \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
5352    \fi}}
```

12.6 Arabic justification

```
5353 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5354 \def\bblar@chars{%
5355  0628,0629,062A,062B,062C,062D,062E,062F,0630,0631,0632,0633,%
5356  0634,0635,0636,0637,0638,0639,063A,063B,063C,063D,063E,063F,%
5357  0640,0641,0642,0643,0644,0645,0646,0647,0649}
5358 \def\bblar@elongated{%
5359  0626,0628,062A,062B,0633,0634,0635,0636,063B,%
5360  063C,063D,063E,063F,0641,0642,0643,0644,0646,%
5361  0649,064A}
5362 \begingroup
5363  \catcode`\_=11 \catcode`\:=11
```

```
5364    \gdef\bblar@nofswarn{\gdef\msg_warning:nnx##1##2##3{}}
5365 \endgroup
5366 \gdef\bbl@arabicjust{%
5367    \let\bbl@arabicjust\relax
5368    \newattribute\bblar@kashida
5369    \directlua{ Babel.attr_kashida = luatexbase.registernumber'bblar@kashida' }%
5370    \bblar@kashida=\z@
5371    \bbl@patchfont{{\bbl@parsejalt}}%
5372    \directlua{
5373      Babel.arabic.elong_map   = Babel.arabic.elong_map or {}
5374      Babel.arabic.elong_map[\the\localeid]   = {}
5375      luatexbase.add_to_callback('post_linebreak_filter',
5376        Babel.arabic.justify, 'Babel.arabic.justify')
5377      luatexbase.add_to_callback('hpack_filter',
5378        Babel.arabic.justify_hbox, 'Babel.arabic.justify_hbox')
5379    }}%
5380 % Save both node lists to make replacement. TODO. Save also widths to
5381 % make computations
5382 \def\bblar@fetchjalt#1#2#3#4{%
5383    \bbl@exp{\\\bbl@foreach{#1}}{%
5384      \bbl@ifunset{bblar@JE@##1}%
5385        {\setbox\z@\hbox{^^^^200d\char"##1#2}}%
5386        {\setbox\z@\hbox{^^^^200d\char"\@nameuse{bblar@JE@##1}#2}}%
5387      \directlua{%
5388        local last = nil
5389        for item in node.traverse(tex.box[0].head) do
5390          if item.id == node.id'glyph' and item.char > 0x600 and
5391              not (item.char == 0x200D) then
5392            last = item
5393          end
5394        end
5395        Babel.arabic.#3['##1#4'] = last.char
5396    }}}
5397 % Brute force. No rules at all, yet. The ideal: look at jalt table. And
5398 % perhaps other tables (falt?, cswh?). What about kaf? And diacritic
5399 % positioning?
5400 \gdef\bbl@parsejalt{%
5401    \ifx\addfontfeature\@undefined\else
5402      \bbl@xin@{/e}{/\bbl@cl{lnbrk}}%
5403      \ifin@
5404        \directlua{%
5405          if Babel.arabic.elong_map[\the\localeid][\fontid\font] == nil then
5406            Babel.arabic.elong_map[\the\localeid][\fontid\font] = {}
5407            tex.print([[\string\csname\space bbl@parsejalti\endcsname]])
5408          end
5409        }%
5410      \fi
5411    \fi}
5412 \gdef\bbl@parsejalti{%
5413    \begingroup
5414      \let\bbl@parsejalt\relax      % To avoid infinite loop
5415      \edef\bbl@tempb{\fontid\font}%
5416      \bblar@nofswarn
5417      \bblar@fetchjalt\bblar@elongated{}{from}{}%
5418      \bblar@fetchjalt\bblar@chars{^^^^064a}{from}{a}% Alef maksura
5419      \bblar@fetchjalt\bblar@chars{^^^^0649}{from}{y}% Yeh
5420      \addfontfeature{RawFeature=+jalt}%
5421      % \@namedef{bblar@JE@0643}{06AA}% todo: catch medial kaf
5422      \bblar@fetchjalt\bblar@elongated{}{dest}{}%
5423      \bblar@fetchjalt\bblar@chars{^^^^064a}{dest}{a}%
5424      \bblar@fetchjalt\bblar@chars{^^^^0649}{dest}{y}%
5425        \directlua{%
5426          for k, v in pairs(Babel.arabic.from) do
```

170

```
5427          if Babel.arabic.dest[k] and
5428              not (Babel.arabic.from[k] == Babel.arabic.dest[k]) then
5429            Babel.arabic.elong_map[\the\localeid][\bbl@tempb]
5430              [Babel.arabic.from[k]] = Babel.arabic.dest[k]
5431          end
5432        end
5433      }%
5434    \endgroup}
5435 %
5436 \begingroup
5437 \catcode`\#=11
5438 \catcode`\~=11
5439 \directlua{
5440
5441 Babel.arabic = Babel.arabic or {}
5442 Babel.arabic.from = {}
5443 Babel.arabic.dest = {}
5444 Babel.arabic.justify_factor = 0.95
5445 Babel.arabic.justify_enabled = true
5446
5447 function Babel.arabic.justify(head)
5448   if not Babel.arabic.justify_enabled then return head end
5449   for line in node.traverse_id(node.id'hlist', head) do
5450     Babel.arabic.justify_hlist(head, line)
5451   end
5452   return head
5453 end
5454
5455 function Babel.arabic.justify_hbox(head, gc, size, pack)
5456   local has_inf = false
5457   if Babel.arabic.justify_enabled and pack == 'exactly' then
5458     for n in node.traverse_id(12, head) do
5459       if n.stretch_order > 0 then has_inf = true end
5460     end
5461     if not has_inf then
5462       Babel.arabic.justify_hlist(head, nil, gc, size, pack)
5463     end
5464   end
5465   return head
5466 end
5467
5468 function Babel.arabic.justify_hlist(head, line, gc, size, pack)
5469   local d, new
5470   local k_list, k_item, pos_inline
5471   local width, width_new, full, k_curr, wt_pos, goal, shift
5472   local subst_done = false
5473   local elong_map = Babel.arabic.elong_map
5474   local last_line
5475   local GLYPH = node.id'glyph'
5476   local KASHIDA = Babel.attr_kashida
5477   local LOCALE = Babel.attr_locale
5478
5479   if line == nil then
5480     line = {}
5481     line.glue_sign = 1
5482     line.glue_order = 0
5483     line.head = head
5484     line.shift = 0
5485     line.width = size
5486   end
5487
5488   % Exclude last line. todo. But-- it discards one-word lines, too!
5489   % ? Look for glue = 12:15
```

```
5490    if (line.glue_sign == 1 and line.glue_order == 0) then
5491      elongs = {}      % Stores elongated candidates of each line
5492      k_list = {}      % And all letters with kashida
5493      pos_inline = 0  % Not yet used
5494
5495      for n in node.traverse_id(GLYPH, line.head) do
5496        pos_inline = pos_inline + 1 % To find where it is. Not used.
5497
5498        % Elongated glyphs
5499        if elong_map then
5500          local locale = node.get_attribute(n, LOCALE)
5501          if elong_map[locale] and elong_map[locale][n.font] and
5502              elong_map[locale][n.font][n.char] then
5503            table.insert(elongs, {node = n, locale = locale} )
5504            node.set_attribute(n.prev, KASHIDA, 0)
5505          end
5506        end
5507
5508        % Tatwil
5509        if Babel.kashida_wts then
5510          local k_wt = node.get_attribute(n, KASHIDA)
5511          if k_wt > 0 then % todo. parameter for multi inserts
5512            table.insert(k_list, {node = n, weight = k_wt, pos = pos_inline})
5513          end
5514        end
5515
5516      end % of node.traverse_id
5517
5518      if #elongs == 0 and #k_list == 0 then goto next_line end
5519      full  = line.width
5520      shift = line.shift
5521      goal  = full * Babel.arabic.justify_factor % A bit crude
5522      width = node.dimensions(line.head)     % The 'natural' width
5523
5524      % == Elongated ==
5525      % Original idea taken from 'chikenize'
5526      while (#elongs > 0 and width < goal) do
5527        subst_done = true
5528        local x = #elongs
5529        local curr = elongs[x].node
5530        local oldchar = curr.char
5531        curr.char = elong_map[elongs[x].locale][curr.font][curr.char]
5532        width = node.dimensions(line.head)  % Check if the line is too wide
5533        % Substitute back if the line would be too wide and break:
5534        if width > goal then
5535          curr.char = oldchar
5536          break
5537        end
5538        % If continue, pop the just substituted node from the list:
5539        table.remove(elongs, x)
5540      end
5541
5542      % == Tatwil ==
5543      if #k_list == 0 then goto next_line end
5544
5545      width = node.dimensions(line.head)     % The 'natural' width
5546      k_curr = #k_list
5547      wt_pos = 1
5548
5549      while width < goal do
5550        subst_done = true
5551        k_item = k_list[k_curr].node
5552        if k_list[k_curr].weight == Babel.kashida_wts[wt_pos] then
```

172

```
5553        d = node.copy(k_item)
5554        d.char = 0x0640
5555        line.head, new = node.insert_after(line.head, k_item, d)
5556        width_new = node.dimensions(line.head)
5557        if width > goal or width == width_new then
5558          node.remove(line.head, new) % Better compute before
5559          break
5560        end
5561        width = width_new
5562      end
5563      if k_curr == 1 then
5564        k_curr = #k_list
5565        wt_pos = (wt_pos >= table.getn(Babel.kashida_wts)) and 1 or wt_pos+1
5566      else
5567        k_curr = k_curr - 1
5568      end
5569    end
5570
5571    ::next_line::
5572
5573    % Must take into account marks and ins, see luatex manual.
5574    % Have to be executed only if there are changes. Investigate
5575    % what's going on exactly.
5576    if subst_done and not gc then
5577      d = node.hpack(line.head, full, 'exactly')
5578      d.shift = shift
5579      node.insert_before(head, line, d)
5580      node.remove(head, line)
5581    end
5582  end % if process line
5583 end
5584 }
5585 \endgroup
5586 \fi\fi % Arabic just block
```

12.7   Common stuff

```
5587 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
5588 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
5589 \DisableBabelHook{babel-fontspec}
5590 ⟨⟨Font selection⟩⟩
```

12.8   Automatic fonts and ids switching

After defining the blocks for a number of scripts (must be extended and very likely fine tuned), we define a short function which just traverse the node list to carry out the replacements. The table loc_to_scr gets the locale form a script range (note the locale is the key, and that there is an intermediate table built on the fly for optimization). This locale is then used to get the \language and the \localeid as stored in locale_props, as well as the font (as requested). In the latter table a key starting with / maps the font from the global one (the key) to the local one (the value). Maths are skipped and discretionaries are handled in a special way.

```
5591 % TODO - to a lua file
5592 \directlua{
5593 Babel.script_blocks = {
5594   ['dflt'] = {},
5595   ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
5596             {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
5597   ['Armn'] = {{0x0530, 0x058F}},
5598   ['Beng'] = {{0x0980, 0x09FF}},
5599   ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
5600   ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
5601   ['Cyrl'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
5602             {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
5603   ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
```

```
5604   ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
5605             {0xAB00, 0xAB2F}},
5606   ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
5607   % Don't follow strictly Unicode, which places some Coptic letters in
5608   % the 'Greek and Coptic' block
5609   ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
5610   ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
5611             {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
5612             {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
5613             {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
5614             {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
5615             {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
5616   ['Hebr'] = {{0x0590, 0x05FF}},
5617   ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
5618             {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
5619   ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
5620   ['Knda'] = {{0x0C80, 0x0CFF}},
5621   ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
5622             {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
5623             {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
5624   ['Laoo'] = {{0x0E80, 0x0EFF}},
5625   ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
5626             {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
5627             {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
5628   ['Mahj'] = {{0x11150, 0x1117F}},
5629   ['Mlym'] = {{0x0D00, 0x0D7F}},
5630   ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
5631   ['Orya'] = {{0x0B00, 0x0B7F}},
5632   ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
5633   ['Syrc'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
5634   ['Taml'] = {{0x0B80, 0x0BFF}},
5635   ['Telu'] = {{0x0C00, 0x0C7F}},
5636   ['Tfng'] = {{0x2D30, 0x2D7F}},
5637   ['Thai'] = {{0x0E00, 0x0E7F}},
5638   ['Tibt'] = {{0x0F00, 0x0FFF}},
5639   ['Vaii'] = {{0xA500, 0xA63F}},
5640   ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
5641 }
5642
5643 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
5644 Babel.script_blocks.Hant = Babel.script_blocks.Hans
5645 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
5646
5647 function Babel.locale_map(head)
5648   if not Babel.locale_mapped then return head end
5649
5650   local LOCALE = Babel.attr_locale
5651   local GLYPH = node.id('glyph')
5652   local inmath = false
5653   local toloc_save
5654   for item in node.traverse(head) do
5655     local toloc
5656     if not inmath and item.id == GLYPH then
5657       % Optimization: build a table with the chars found
5658       if Babel.chr_to_loc[item.char] then
5659         toloc = Babel.chr_to_loc[item.char]
5660       else
5661         for lc, maps in pairs(Babel.loc_to_scr) do
5662           for _, rg in pairs(maps) do
5663             if item.char >= rg[1] and item.char <= rg[2] then
5664               Babel.chr_to_loc[item.char] = lc
5665               toloc = lc
5666               break
```

174

```
5667              end
5668            end
5669          end
5670        end
5671        % Now, take action, but treat composite chars in a different
5672        % fashion, because they 'inherit' the previous locale. Not yet
5673        % optimized.
5674        if not toloc and
5675            (item.char >= 0x0300 and item.char <= 0x036F) or
5676            (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
5677            (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
5678          toloc = toloc_save
5679        end
5680        if toloc and toloc > -1 then
5681          if Babel.locale_props[toloc].lg then
5682            item.lang = Babel.locale_props[toloc].lg
5683            node.set_attribute(item, LOCALE, toloc)
5684          end
5685          if Babel.locale_props[toloc]['/'..item.font] then
5686            item.font = Babel.locale_props[toloc]['/'..item.font]
5687          end
5688          toloc_save = toloc
5689        end
5690      elseif not inmath and item.id == 7 then
5691        item.replace = item.replace and Babel.locale_map(item.replace)
5692        item.pre     = item.pre and Babel.locale_map(item.pre)
5693        item.post    = item.post and Babel.locale_map(item.post)
5694      elseif item.id == node.id'math' then
5695        inmath = (item.subtype == 0)
5696      end
5697    end
5698    return head
5699 end
5700 }
```

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

```
5701 \newcommand\babelcharproperty[1]{%
5702   \count@=#1\relax
5703   \ifvmode
5704     \expandafter\bbl@chprop
5705   \else
5706     \bbl@error{\string\babelcharproperty\space can be used only in\\%
5707                vertical mode (preamble or between paragraphs)}%
5708                {See the manual for futher info}%
5709   \fi}
5710 \newcommand\bbl@chprop[3][\the\count@]{%
5711   \@tempcnta=#1\relax
5712   \bbl@ifunset{bbl@chprop@#2}%
5713     {\bbl@error{No property named '#2'. Allowed values are\\%
5714                direction (bc), mirror (bmg), and linebreak (lb)}%
5715                {See the manual for futher info}}%
5716     {}%
5717   \loop
5718     \bbl@cs{chprop@#2}{#3}%
5719   \ifnum\count@<\@tempcnta
5720     \advance\count@\@ne
5721   \repeat}
5722 \def\bbl@chprop@direction#1{%
5723   \directlua{
5724     Babel.characters[\the\count@] =  Babel.characters[\the\count@] or {}
5725     Babel.characters[\the\count@]['d'] = '#1'
5726   }}
```

175

```
5727 \let\bbl@chprop@bc\bbl@chprop@direction
5728 \def\bbl@chprop@mirror#1{%
5729   \directlua{
5730     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5731     Babel.characters[\the\count@]['m'] = '\number#1'
5732   }}
5733 \let\bbl@chprop@bmg\bbl@chprop@mirror
5734 \def\bbl@chprop@linebreak#1{%
5735   \directlua{
5736     Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
5737     Babel.cjk_characters[\the\count@]['c'] = '#1'
5738   }}
5739 \let\bbl@chprop@lb\bbl@chprop@linebreak
5740 \def\bbl@chprop@locale#1{%
5741   \directlua{
5742     Babel.chr_to_loc = Babel.chr_to_loc or {}
5743     Babel.chr_to_loc[\the\count@] =
5744       \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@@#1}}\space
5745   }}
```

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.

```
5746 \directlua{
5747   Babel.nohyphenation = \the\l@nohyphenation
5748 }
```

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).

```
5749 \begingroup
5750 \catcode`\~=12
5751 \catcode`\%=12
5752 \catcode`\&=14
5753 \catcode`\|=12
5754 \gdef\babelprehyphenation{&%
5755   \@ifnextchar[{\bbl@settransform{0}}{\bbl@settransform{0}[]}}
5756 \gdef\babelposthyphenation{&%
5757   \@ifnextchar[{\bbl@settransform{1}}{\bbl@settransform{1}[]}}
5758 \gdef\bbl@settransform#1[#2]#3#4#5{&%
5759   \ifcase#1
5760     \bbl@activateprehyphen
5761   \else
5762     \bbl@activateposthyphen
5763   \fi
5764   \begingroup
5765     \def\babeltempa{\bbl@add@list\babeltempb}&%
5766     \let\babeltempb\@empty
5767     \def\bbl@tempa{#5}&%
5768     \bbl@replace\bbl@tempa{,}{ ,}&% TODO. Ugly trick to preserve {}
5769     \expandafter\bbl@foreach\expandafter{\bbl@tempa}{&%
5770       \bbl@ifsamestring{##1}{remove}&%
5771         {\bbl@add@list\babeltempb{nil}}&%
5772         {\directlua{
5773            local rep = [=[##1]=]
5774            rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
5775            rep = rep:gsub('^%s*(insert)%s*,', 'insert = true, ')
5776            rep = rep:gsub('(string)%s*=%s*([^%s,]*)', Babel.capture_func)
5777            if #1 == 0 then
```

176

```
5778          rep = rep:gsub('(space)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
5779                'space = {' .. '%2, %3, %4' .. '}')
5780          rep = rep:gsub('(spacefactor)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
5781                'spacefactor = {' .. '%2, %3, %4' .. '}')
5782          rep = rep:gsub('(kashida)%s*=%s*([^%s,]*)', Babel.capture_kashida)
5783        else
5784          rep = rep:gsub(   '(no)%s*=%s*([^%s,]*)', Babel.capture_func)
5785          rep = rep:gsub(  '(pre)%s*=%s*([^%s,]*)', Babel.capture_func)
5786          rep = rep:gsub( '(post)%s*=%s*([^%s,]*)', Babel.capture_func)
5787        end
5788        tex.print([[\string\babeltempa{{]] .. rep .. [[}}]])
5789      }}}&%
5790    \let\bbl@kv@attribute\relax
5791    \let\bbl@kv@label\relax
5792    \bbl@forkv{#2}{\bbl@csarg\edef{kv@##1}{##2}}&%
5793    \ifx\bbl@kv@attribute\relax\else
5794      \edef\bbl@kv@attribute{\expandafter\bbl@stripslash\bbl@kv@attribute}&%
5795    \fi
5796    \directlua{
5797      local lbkr = Babel.linebreaking.replacements[#1]
5798      local u = unicode.utf8
5799      local id, attr, label
5800      if #1 == 0 then
5801        id = \the\csname bbl@id@@#3\endcsname\space
5802      else
5803        id = \the\csname l@#3\endcsname\space
5804      end
5805      \ifx\bbl@kv@attribute\relax
5806        attr = -1
5807      \else
5808        attr = luatexbase.registernumber'\bbl@kv@attribute'
5809      \fi
5810      \ifx\bbl@kv@label\relax\else  &% Same refs:
5811        label = [==[\bbl@kv@label]==]
5812      \fi
5813      &% Convert pattern:
5814      local patt = string.gsub([==[#4]==], '%s', '')
5815      if #1 == 0 then
5816        patt = string.gsub(patt, '|', ' ')
5817      end
5818      if not u.find(patt, '()', nil, true) then
5819        patt = '()' .. patt .. '()'
5820      end
5821      if #1 == 1 then
5822        patt = string.gsub(patt, '%(%)%^', '^()')
5823        patt = string.gsub(patt, '%$%(%)', '()$')
5824      end
5825      patt = u.gsub(patt, '{(.)}',
5826              function (n)
5827                return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
5828              end)
5829      patt = u.gsub(patt, '{(%x%x%x%x+)}',
5830              function (n)
5831                return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%%1')
5832              end)
5833      lbkr[id] = lbkr[id] or {}
5834      table.insert(lbkr[id],
5835        { label=label, attr=attr, pattern=patt, replace={\babeltempb} })
5836    }&%
5837  \endgroup}
5838 \endgroup
5839 \def\bbl@activateposthyphen{%
5840  \let\bbl@activateposthyphen\relax
```

```
5841  \directlua{
5842    require('babel-transforms.lua')
5843    Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
5844  }}
5845 \def\bbl@activateprehyphen{%
5846   \let\bbl@activateprehyphen\relax
5847   \directlua{
5848     require('babel-transforms.lua')
5849     Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
5850  }}
```

12.9   Bidi

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

```
5851 \def\bbl@activate@preotf{%
5852   \let\bbl@activate@preotf\relax  % only once
5853   \directlua{
5854     Babel = Babel or {}
5855     %
5856     function Babel.pre_otfload_v(head)
5857       if Babel.numbers and Babel.digits_mapped then
5858         head = Babel.numbers(head)
5859       end
5860       if Babel.bidi_enabled then
5861         head = Babel.bidi(head, false, dir)
5862       end
5863       return head
5864     end
5865     %
5866     function Babel.pre_otfload_h(head, gc, sz, pt, dir)
5867       if Babel.numbers and Babel.digits_mapped then
5868         head = Babel.numbers(head)
5869       end
5870       if Babel.bidi_enabled then
5871         head = Babel.bidi(head, false, dir)
5872       end
5873       return head
5874     end
5875     %
5876     luatexbase.add_to_callback('pre_linebreak_filter',
5877       Babel.pre_otfload_v,
5878       'Babel.pre_otfload_v',
5879       luatexbase.priority_in_callback('pre_linebreak_filter',
5880         'luaotfload.node_processor') or nil)
5881     %
5882     luatexbase.add_to_callback('hpack_filter',
5883       Babel.pre_otfload_h,
5884       'Babel.pre_otfload_h',
5885       luatexbase.priority_in_callback('hpack_filter',
5886         'luaotfload.node_processor') or nil)
5887  }}
```

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=.

```
5888 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5889   \let\bbl@beforeforeign\leavevmode
5890   \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
5891   \RequirePackage{luatexbase}
5892   \bbl@activate@preotf
5893   \directlua{
```

```
5894        require('babel-data-bidi.lua')
5895        \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
5896          require('babel-bidi-basic.lua')
5897        \or
5898          require('babel-bidi-basic-r.lua')
5899        \fi}
5900      % TODO - to locale_props, not as separate attribute
5901      \newattribute\bbl@attr@dir
5902      \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
5903      % TODO. I don't like it, hackish:
5904      \bbl@exp{\output{\bodydir\pagedir\the\output}}
5905      \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
5906  \fi\fi
5907  \chardef\bbl@thetextdir\z@
5908  \chardef\bbl@thepardir\z@
5909  \def\bbl@getluadir#1{%
5910    \directlua{
5911      if tex.#1dir == 'TLT' then
5912        tex.sprint('0')
5913      elseif tex.#1dir == 'TRT' then
5914        tex.sprint('1')
5915      end}}
5916  \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
5917    \ifcase#3\relax
5918      \ifcase\bbl@getluadir{#1}\relax\else
5919        #2 TLT\relax
5920      \fi
5921    \else
5922      \ifcase\bbl@getluadir{#1}\relax
5923        #2 TRT\relax
5924      \fi
5925    \fi}
5926  \def\bbl@thedir{0}
5927  \def\bbl@textdir#1{%
5928    \bbl@setluadir{text}\textdir{#1}%
5929    \chardef\bbl@thetextdir#1\relax
5930    \edef\bbl@thedir{\the\numexpr\bbl@thepardir*3+#1}%
5931    \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*3+#1}}
5932  \def\bbl@pardir#1{%
5933    \bbl@setluadir{par}\pardir{#1}%
5934    \chardef\bbl@thepardir#1\relax}
5935  \def\bbl@bodydir{\bbl@setluadir{body}\bodydir}
5936  \def\bbl@pagedir{\bbl@setluadir{page}\pagedir}
5937  \def\bbl@dirparastext{\pardir\the\textdir\relax}%    %%%%
5938  %
5939  \ifnum\bbl@bidimode>\z@
5940    \def\bbl@insidemath{0}%
5941    \def\bbl@everymath{\def\bbl@insidemath{1}}
5942    \def\bbl@everydisplay{\def\bbl@insidemath{2}}
5943    \frozen@everymath\expandafter{%
5944      \expandafter\bbl@everymath\the\frozen@everymath}
5945    \frozen@everydisplay\expandafter{%
5946      \expandafter\bbl@everydisplay\the\frozen@everydisplay}
5947    \AtBeginDocument{
5948      \directlua{
5949        function Babel.math_box_dir(head)
5950          if not (token.get_macro('bbl@insidemath') == '0') then
5951            if Babel.hlist_has_bidi(head) then
5952              local d = node.new(node.id'dir')
5953              d.dir = '+TRT'
5954              node.insert_before(head, node.has_glyph(head), d)
5955              for item in node.traverse(head) do
5956                node.set_attribute(item,
```

```
5957                   Babel.attr_dir, token.get_macro('bbl@thedir'))
5958             end
5959           end
5960         end
5961       return head
5962     end
5963     luatexbase.add_to_callback("hpack_filter", Babel.math_box_dir,
5964       "Babel.math_box_dir", 0)
5965  }}%
5966 \fi
```

12.10 Layout

Unlike xetex, luatex requires only minimal changes for right-to-left layouts, particularly in monolingual documents (the engine itself reverses boxes – including column order or headings –, margins, etc.) with bidi=basic, without having to patch almost any macro where text direction is relevant.

\@hangfrom is useful in many contexts and it is redefined always with the layout option.

There are, however, a number of issues when the text direction is not the same as the box direction (as set by \bodydir), and when \parbox and \hangindent are involved. Fortunately, latest releases of luatex simplify a lot the solution with \shapemode.

With the issue #15 I realized commands are best patched, instead of redefined. With a few lines, a modification could be applied to several classes and packages. Now, tabular seems to work (at least in simple cases) with array, tabularx, hhline, colortbl, longtable, booktabs, etc. However, dcolumn still fails.

```
5967 \bbl@trace{Redefinitions for bidi layout}
5968 %
5969 ⟨⟨*More package options⟩⟩ ≡
5970 \chardef\bbl@eqnpos\z@
5971 \DeclareOption{leqno}{\chardef\bbl@eqnpos\@ne}
5972 \DeclareOption{fleqn}{\chardef\bbl@eqnpos\tw@}
5973 ⟨⟨/More package options⟩⟩
5974 %
5975 \def\BabelNoAMSMath{\let\bbl@noamsmath\relax}
5976 \ifnum\bbl@bidimode>\z@
5977   \ifx\matheqdirmode\@undefined\else
5978     \matheqdirmode\@ne
5979   \fi
5980   \let\bbl@eqnodir\relax
5981   \def\bbl@eqdel{()}
5982   \def\bbl@eqnum{%
5983     {\normalfont\normalcolor
5984       \expandafter\@firstoftwo\bbl@eqdel
5985       \theequation
5986       \expandafter\@secondoftwo\bbl@eqdel}}
5987   \def\bbl@puteqno#1{\eqno\hbox{#1}}
5988   \def\bbl@putleqno#1{\leqno\hbox{#1}}
5989   \def\bbl@eqno@flip#1{%
5990     \ifdim\predisplaysize=-\maxdimen
5991       \eqno
5992       \hb@xt@.01pt{\hb@xt@\displaywidth{\hss{#1}}\hss}%
5993     \else
5994       \leqno\hbox{#1}%
5995     \fi}
5996   \def\bbl@leqno@flip#1{%
5997     \ifdim\predisplaysize=-\maxdimen
5998       \leqno
5999       \hb@xt@.01pt{\hss\hb@xt@\displaywidth{{#1}\hss}}%
6000     \else
6001       \eqno\hbox{#1}%
6002     \fi}
6003   \AtBeginDocument{%
6004     \ifx\maketag@@@\@undefined % Normal equation, eqnarray
```

```
6005        \AddToHook{env/equation/begin}{%
6006          \ifnum\bbl@thetextdir>\z@
6007            \let\@eqnnum\bbl@eqnum
6008            \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6009            \chardef\bbl@thetextdir\z@
6010            \bbl@add\normalfont{\bbl@eqnodir}%
6011            \ifcase\bbl@eqnpos
6012              \let\bbl@puteqno\bbl@eqno@flip
6013            \or
6014              \let\bbl@puteqno\bbl@leqno@flip
6015            \fi
6016          \fi}%
6017        \ifnum\bbl@eqnpos=\tw@\else
6018          \def\endequation{\bbl@puteqno{\@eqnnum}$$\@ignoretrue}%
6019        \fi
6020        \AddToHook{env/eqnarray/begin}{%
6021          \ifnum\bbl@thetextdir>\z@
6022            \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6023            \chardef\bbl@thetextdir\z@
6024            \bbl@add\normalfont{\bbl@eqnodir}%
6025            \ifnum\bbl@eqnpos=\@ne
6026              \def\@eqnnum{%
6027               \setbox\z@\hbox{\bbl@eqnum}%
6028               \hbox to0.01pt{\hss\hbox to\displaywidth{\box\z@\hss}}}%
6029            \else
6030               \let\@eqnnum\bbl@eqnum
6031            \fi
6032          \fi}
6033        % Hack. YA luatex bug?:
6034        \expandafter\bbl@sreplace\csname] \endcsname{$$}{\eqno\kern.001pt$$}%
6035      \else % amstex
6036        \ifx\bbl@noamsmath\@undefined
6037          \ifnum\bbl@eqnpos=\@ne
6038            \let\bbl@ams@lap\hbox
6039          \else
6040            \let\bbl@ams@lap\llap
6041          \fi
6042          \ExplSyntaxOn
6043          \bbl@sreplace\intertext@{\normalbaselines}%
6044            {\normalbaselines
6045             \ifx\bbl@eqnodir\relax\else\bbl@pardir\@ne\bbl@eqnodir\fi}%
6046          \ExplSyntaxOff
6047          \def\bbl@ams@tagbox#1#2{#1{\bbl@eqnodir#2}}% #1=hbox|@lap|flip
6048          \ifx\bbl@ams@lap\hbox % leqno
6049            \def\bbl@ams@flip#1{%
6050              \hbox to 0.01pt{\hss\hbox to\displaywidth{{#1}\hss}}}%
6051          \else % eqno
6052            \def\bbl@ams@flip#1{%
6053              \hbox to 0.01pt{\hbox to\displaywidth{\hss#1}\hss}}%
6054          \fi
6055          \def\bbl@ams@preset#1{%
6056            \ifnum\bbl@thetextdir>\z@
6057              \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6058              \bbl@sreplace\textdef@{\hbox}{\bbl@ams@tagbox\hbox}%
6059              \bbl@sreplace\maketag@@@{\hbox}{\bbl@ams@tagbox#1}%
6060            \fi}%
6061          \ifnum\bbl@eqnpos=\tw@\else
6062            \def\bbl@ams@equation{%
6063              \ifnum\bbl@thetextdir>\z@
6064                \edef\bbl@eqnodir{\noexpand\bbl@textdir{\the\bbl@thetextdir}}%
6065                \chardef\bbl@thetextdir\z@
6066                \bbl@add\normalfont{\bbl@eqnodir}%
6067                \ifcase\bbl@eqnpos
```

```
6068              \def\veqno##1##2{\bbl@eqno@flip{##1##2}}%
6069            \or
6070              \def\veqno##1##2{\bbl@leqno@flip{##1##2}}%
6071           \fi
6072         \fi}%
6073       \AddToHook{env/equation/begin}{\bbl@ams@equation}%
6074       \AddToHook{env/equation*/begin}{\bbl@ams@equation}%
6075     \fi
6076     \AddToHook{env/cases/begin}{\bbl@ams@preset\bbl@ams@lap}%
6077     \AddToHook{env/multline/begin}{\bbl@ams@preset\hbox}%
6078     \AddToHook{env/gather/begin}{\bbl@ams@preset\bbl@ams@lap}%
6079     \AddToHook{env/gather*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6080     \AddToHook{env/align/begin}{\bbl@ams@preset\bbl@ams@lap}%
6081     \AddToHook{env/align*/begin}{\bbl@ams@preset\bbl@ams@lap}%
6082     \AddToHook{env/eqnalign/begin}{\bbl@ams@preset\hbox}%
6083     % Hackish, for proper alignment. Don't ask me why it works!:
6084     \bbl@exp{% Avoid a 'visible' conditional
6085       \\\AddToHook{env/align*/end}{\<iftag@>\<else>\\\tag*{}\<fi>}}%
6086     \AddToHook{env/flalign/begin}{\bbl@ams@preset\hbox}%
6087     \AddToHook{env/split/before}{%
6088       \ifnum\bbl@thetextdir>\z@
6089         \bbl@ifsamestring\@currenvir{equation}%
6090           {\ifx\bbl@ams@lap\hbox % leqno
6091              \def\bbl@ams@flip#1{%
6092                \hbox to 0.01pt{\hbox to\displaywidth{{#1}\hss}\hss}}%
6093            \else
6094              \def\bbl@ams@flip#1{%
6095                \hbox to 0.01pt{\hss\hbox to\displaywidth{\hss{#1}}}}%
6096            \fi}%
6097          {}%
6098       \fi}%
6099     \fi
6100   \fi}
6101 \fi
6102 \ifx\bbl@opt@layout\@nnil\endinput\fi  % if no layout
6103 \ifnum\bbl@bidimode>\z@
6104   \def\bbl@nextfake#1{%  non-local changes, use always inside a group!
6105     \bbl@exp{%
6106       \def\\\bbl@insidemath{0}%
6107       \mathdir\the\bodydir
6108       #1%               Once entered in math, set boxes to restore values
6109       \<ifmmode>%
6110         \everyvbox{%
6111           \the\everyvbox
6112           \bodydir\the\bodydir
6113           \mathdir\the\mathdir
6114           \everyhbox{\the\everyhbox}%
6115           \everyvbox{\the\everyvbox}}%
6116         \everyhbox{%
6117           \the\everyhbox
6118           \bodydir\the\bodydir
6119           \mathdir\the\mathdir
6120           \everyhbox{\the\everyhbox}%
6121           \everyvbox{\the\everyvbox}}%
6122       \<fi>}}%
6123   \def\@hangfrom#1{%
6124     \setbox\@tempboxa\hbox{{#1}}%
6125     \hangindent\wd\@tempboxa
6126     \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
6127       \shapemode\@ne
6128     \fi
6129     \noindent\box\@tempboxa}
6130 \fi
```

```
6131 \IfBabelLayout{tabular}
6132   {\let\bbl@OL@@tabular\@tabular
6133    \bbl@replace\@tabular{$}{\bbl@nextfake$}%
6134    \let\bbl@NL@@tabular\@tabular
6135    \AtBeginDocument{%
6136      \ifx\bbl@NL@@tabular\@tabular\else
6137        \bbl@replace\@tabular{$}{\bbl@nextfake$}%
6138        \let\bbl@NL@@tabular\@tabular
6139      \fi}}
6140   {}
6141 \IfBabelLayout{lists}
6142   {\let\bbl@OL@list\list
6143    \bbl@sreplace\list{\parshape}{\bbl@listparshape}%
6144    \let\bbl@NL@list\list
6145    \def\bbl@listparshape#1#2#3{%
6146      \parshape #1 #2 #3 %
6147      \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
6148        \shapemode\tw@
6149      \fi}}
6150   {}
6151 \IfBabelLayout{graphics}
6152   {\let\bbl@pictresetdir\relax
6153    \def\bbl@pictsetdir#1{%
6154      \ifcase\bbl@thetextdir
6155        \let\bbl@pictresetdir\relax
6156      \else
6157        \ifcase#1\bodydir TLT  % Remember this sets the inner boxes
6158          \or\textdir TLT
6159          \else\bodydir TLT \textdir TLT
6160        \fi
6161        % \(text|par)dir required in pgf:
6162        \def\bbl@pictresetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}%
6163      \fi}%
6164    \AddToHook{env/picture/begin}{\bbl@pictsetdir\tw@}%
6165    \directlua{
6166      Babel.get_picture_dir = true
6167      Babel.picture_has_bidi = 0
6168      %
6169      function Babel.picture_dir (head)
6170        if not Babel.get_picture_dir then return head end
6171        if Babel.hlist_has_bidi(head) then
6172          Babel.picture_has_bidi = 1
6173        end
6174        return head
6175      end
6176      luatexbase.add_to_callback("hpack_filter", Babel.picture_dir,
6177        "Babel.picture_dir")
6178    }%
6179    \AtBeginDocument{%
6180      \long\def\put(#1,#2)#3{%
6181        \@killglue
6182        % Try:
6183        \ifx\bbl@pictresetdir\relax
6184          \def\bbl@tempc{0}%
6185        \else
6186          \directlua{
6187            Babel.get_picture_dir = true
6188            Babel.picture_has_bidi = 0
6189          }%
6190          \setbox\z@\hb@xt@\z@{%
6191            \@defaultunitsset\@tempdimc{#1}\unitlength
6192            \kern\@tempdimc
6193            #3\hss}% TODO: #3 executed twice (below). That's bad.
```

```
6194          \edef\bbl@tempc{\directlua{tex.print(Babel.picture_has_bidi)}}}%
6195        \fi
6196        % Do:
6197        \@defaultunitsset\@tempdimc{#2}\unitlength
6198        \raise\@tempdimc\hb@xt@\z@{%
6199          \@defaultunitsset\@tempdimc{#1}\unitlength
6200          \kern\@tempdimc
6201          {\ifnum\bbl@tempc>\z@\bbl@pictresetdir\fi#3}\hss}%
6202        \ignorespaces}%
6203      \MakeRobust\put}%
6204    \AtBeginDocument
6205      {\AddToHook{cmd/diagbox@pict/before}{\let\bbl@pictsetdir\@gobble}%
6206      \ifx\pgfpicture\@undefined\else % TODO. Allow deactivate?
6207        \AddToHook{env/pgfpicture/begin}{\bbl@pictsetdir\@ne}%
6208        \bbl@add\pgfinterruptpicture{\bbl@pictresetdir}%
6209        \bbl@add\pgfsys@beginpicture{\bbl@pictsetdir\z@}%
6210      \fi
6211      \ifx\tikzpicture\@undefined\else
6212        \AddToHook{env/tikzpicture/begin}{\bbl@pictsetdir\z@}%
6213        \bbl@add\tikz@atbegin@node{\bbl@pictresetdir}%
6214        \bbl@sreplace\tikz{\begingroup}{\begingroup\bbl@pictsetdir\tw@}%
6215      \fi
6216      \ifx\tcolorbox\@undefined\else
6217        \AddToHook{env/tcolorbox/begin}{\bbl@pictsetdir\@ne}%
6218        \bbl@sreplace\tcb@savebox
6219          {\ignorespaces}{\ignorespaces\bbl@pictresetdir}%
6220        \ifx\tikzpicture@tcb@hooked\@undefined\else
6221          \bbl@sreplace\tikzpicture@tcb@hooked{\noexpand\tikzpicture}%
6222            {\textdir TLT\noexpand\tikzpicture}%
6223        \fi
6224      \fi
6225    }}
6226    {}
```

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.

```
6227 \IfBabelLayout{counters}%
6228   {\let\bbl@OL@@textsuperscript\@textsuperscript
6229    \bbl@sreplace\@textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
6230    \let\bbl@latinarabic=\@arabic
6231    \let\bbl@OL@@arabic\@arabic
6232    \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
6233    \@ifpackagewith{babel}{bidi=default}%
6234      {\let\bbl@asciiroman=\@roman
6235       \let\bbl@OL@@roman\@roman
6236       \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciiroman#1}}}%
6237       \let\bbl@asciiRoman=\@Roman
6238       \let\bbl@OL@@roman\@Roman
6239       \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}%
6240       \let\bbl@OL@labelenumii\labelenumii
6241       \def\labelenumii{)\theenumii(}%
6242       \let\bbl@OL@p@enumiii\p@enumiii
6243       \def\p@enumiii{\p@enumii)\theenumii(}}{}}{}
6244 ⟨⟨Footnote changes⟩⟩
6245 \IfBabelLayout{footnotes}%
6246   {\let\bbl@OL@footnote\footnote
6247    \BabelFootnote\footnote\languagename{}{}%
6248    \BabelFootnote\localfootnote\languagename{}{}%
6249    \BabelFootnote\mainfootnote{}{}{}}
6250   {}
```

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.

```
6251 \IfBabelLayout{extras}%
6252   {\let\bbl@OL@underline\underline
6253    \bbl@sreplace\underline{$\@@underline}{\bbl@nextfake$\@@underline}%
6254    \let\bbl@OL@LaTeX2e\LaTeX2e
6255    \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
6256      \if b\expandafter\@car\f@series\@nil\boldmath\fi
6257      \babelsublr{%
6258        \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}}}
6259   {}
6260 ⟨/luatex⟩
```

12.11 Lua: transforms

After declaring the table containing the patterns with their replacements, we define some auxiliary functions: `str_to_nodes` converts the string returned by a function to a node list, taking the node at `base` as a model (font, language, etc.); `fetch_word` fetches a series of glyphs and discretionaries, which `pattern` is matched against (if there is a match, it is called again before trying other patterns, and this is very likely the main bottleneck).

`post_hyphenate_replace` is the callback applied after `lang.hyphenate`. This means the automatic hyphenation points are known. As empty captures return a byte position (as explained in the luatex manual), we must convert it to a utf8 position. With `first`, the last byte can be the leading byte in a utf8 sequence, so we just remove it and add 1 to the resulting length. With `last` we must take into account the capture position points to the next character. Here `word_head` points to the starting node of the text to be matched.

```
6261 ⟨*transforms⟩
6262 Babel.linebreaking.replacements = {}
6263 Babel.linebreaking.replacements[0] = {}  -- pre
6264 Babel.linebreaking.replacements[1] = {}  -- post
6265
6266 -- Discretionaries contain strings as nodes
6267 function Babel.str_to_nodes(fn, matches, base)
6268   local n, head, last
6269   if fn == nil then return nil end
6270   for s in string.utfvalues(fn(matches)) do
6271     if base.id == 7 then
6272       base = base.replace
6273     end
6274     n = node.copy(base)
6275     n.char    = s
6276     if not head then
6277       head = n
6278     else
6279       last.next = n
6280     end
6281     last = n
6282   end
6283   return head
6284 end
6285
6286 Babel.fetch_subtext = {}
6287
6288 Babel.ignore_pre_char = function(node)
6289   return (node.lang == Babel.nohyphenation)
6290 end
6291
6292 -- Merging both functions doesn't seen feasible, because there are too
6293 -- many differences.
6294 Babel.fetch_subtext[0] = function(head)
6295   local word_string = ''
6296   local word_nodes = {}
6297   local lang
6298   local item = head
6299   local inmath = false
```

```
6300
6301    while item do
6302
6303      if item.id == 11 then
6304        inmath = (item.subtype == 0)
6305      end
6306
6307      if inmath then
6308        -- pass
6309
6310      elseif item.id == 29 then
6311        local locale = node.get_attribute(item, Babel.attr_locale)
6312
6313        if lang == locale or lang == nil then
6314          lang = lang or locale
6315          if Babel.ignore_pre_char(item) then
6316            word_string = word_string .. Babel.us_char
6317          else
6318            word_string = word_string .. unicode.utf8.char(item.char)
6319          end
6320          word_nodes[#word_nodes+1] = item
6321        else
6322          break
6323        end
6324
6325      elseif item.id == 12 and item.subtype == 13 then
6326        word_string = word_string .. ' '
6327        word_nodes[#word_nodes+1] = item
6328
6329      -- Ignore leading unrecognized nodes, too.
6330      elseif word_string ~= '' then
6331        word_string = word_string .. Babel.us_char
6332        word_nodes[#word_nodes+1] = item  -- Will be ignored
6333      end
6334
6335      item = item.next
6336    end
6337
6338    -- Here and above we remove some trailing chars but not the
6339    -- corresponding nodes. But they aren't accessed.
6340    if word_string:sub(-1) == ' ' then
6341      word_string = word_string:sub(1,-2)
6342    end
6343    word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
6344    return word_string, word_nodes, item, lang
6345 end
6346
6347 Babel.fetch_subtext[1] = function(head)
6348    local word_string = ''
6349    local word_nodes = {}
6350    local lang
6351    local item = head
6352    local inmath = false
6353
6354    while item do
6355
6356      if item.id == 11 then
6357        inmath = (item.subtype == 0)
6358      end
6359
6360      if inmath then
6361        -- pass
6362
```

```
6363        elseif item.id == 29 then
6364          if item.lang == lang or lang == nil then
6365            if (item.char ~= 124) and (item.char ~= 61) then -- not =, not |
6366              lang = lang or item.lang
6367              word_string = word_string .. unicode.utf8.char(item.char)
6368              word_nodes[#word_nodes+1] = item
6369            end
6370          else
6371            break
6372          end
6373
6374        elseif item.id == 7 and item.subtype == 2 then
6375          word_string = word_string .. '='
6376          word_nodes[#word_nodes+1] = item
6377
6378        elseif item.id == 7 and item.subtype == 3 then
6379          word_string = word_string .. '|'
6380          word_nodes[#word_nodes+1] = item
6381
6382        -- (1) Go to next word if nothing was found, and (2) implicitly
6383        -- remove leading USs.
6384        elseif word_string == '' then
6385          -- pass
6386
6387        -- This is the responsible for splitting by words.
6388        elseif (item.id == 12 and item.subtype == 13) then
6389          break
6390
6391        else
6392          word_string = word_string .. Babel.us_char
6393          word_nodes[#word_nodes+1] = item  -- Will be ignored
6394        end
6395
6396      item = item.next
6397    end
6398
6399    word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
6400    return word_string, word_nodes, item, lang
6401 end
6402
6403 function Babel.pre_hyphenate_replace(head)
6404   Babel.hyphenate_replace(head, 0)
6405 end
6406
6407 function Babel.post_hyphenate_replace(head)
6408   Babel.hyphenate_replace(head, 1)
6409 end
6410
6411 Babel.us_char = string.char(31)
6412
6413 function Babel.hyphenate_replace(head, mode)
6414   local u = unicode.utf8
6415   local lbkr = Babel.linebreaking.replacements[mode]
6416
6417   local word_head = head
6418
6419   while true do  -- for each subtext block
6420
6421     local w, w_nodes, nw, lang = Babel.fetch_subtext[mode](word_head)
6422
6423     if Babel.debug then
6424       print()
6425       print((mode == 0) and '@@@@<' or '@@@@>', w)
```

```
6426      end

6427

6428      if nw == nil and w == '' then break end

6429

6430      if not lang then goto next end
6431      if not lbkr[lang] then goto next end

6432

6433      -- For each saved (pre|post)hyphenation. TODO. Reconsider how
6434      -- loops are nested.
6435      for k=1, #lbkr[lang] do
6436        local p = lbkr[lang][k].pattern
6437        local r = lbkr[lang][k].replace
6438        local attr = lbkr[lang][k].attr or -1

6439

6440        if Babel.debug then
6441          print('*****', p, mode)
6442        end

6443

6444        -- This variable is set in some cases below to the first *byte*
6445        -- after the match, either as found by u.match (faster) or the
6446        -- computed position based on sc if w has changed.
6447        local last_match = 0
6448        local step = 0

6449

6450        -- For every match.
6451        while true do
6452          if Babel.debug then
6453            print('=====')
6454          end
6455          local new  -- used when inserting and removing nodes

6456

6457          local matches = { u.match(w, p, last_match) }

6458

6459          if #matches < 2 then break end

6460

6461          -- Get and remove empty captures (with ()'s, which return a
6462          -- number with the position), and keep actual captures
6463          -- (from (...)), if any, in matches.
6464          local first = table.remove(matches, 1)
6465          local last  = table.remove(matches, #matches)
6466          -- Non re-fetched substrings may contain \31, which separates
6467          -- subsubstrings.
6468          if string.find(w:sub(first, last-1), Babel.us_char) then break end

6469

6470          local save_last = last -- with A()BC()D, points to D

6471

6472          -- Fix offsets, from bytes to unicode. Explained above.
6473          first = u.len(w:sub(1, first-1)) + 1
6474          last  = u.len(w:sub(1, last-1)) -- now last points to C

6475

6476          -- This loop stores in a small table the nodes
6477          -- corresponding to the pattern. Used by 'data' to provide a
6478          -- predictable behavior with 'insert' (w_nodes is modified on
6479          -- the fly), and also access to 'remove'd nodes.
6480          local sc = first-1          -- Used below, too
6481          local data_nodes = {}

6482

6483          local enabled = true
6484          for q = 1, last-first+1 do
6485            data_nodes[q] = w_nodes[sc+q]
6486            if enabled
6487                and attr > -1
6488                and not node.has_attribute(data_nodes[q], attr)
```

```
6489                then
6490                  enabled = false
6491                end
6492            end
6493
6494            -- This loop traverses the matched substring and takes the
6495            -- corresponding action stored in the replacement list.
6496            -- sc = the position in substr nodes / string
6497            -- rc = the replacement table index
6498            local rc = 0
6499
6500            while rc < last-first+1 do -- for each replacement
6501              if Babel.debug then
6502                print('.....', rc + 1)
6503              end
6504              sc = sc + 1
6505              rc = rc + 1
6506
6507              if Babel.debug then
6508                Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6509                local ss = ''
6510                for itt in node.traverse(head) do
6511                  if itt.id == 29 then
6512                    ss = ss .. unicode.utf8.char(itt.char)
6513                  else
6514                    ss = ss .. '{' .. itt.id .. '}'
6515                  end
6516                end
6517                print('*****************', ss)
6518
6519              end
6520
6521              local crep = r[rc]
6522              local item = w_nodes[sc]
6523              local item_base = item
6524              local placeholder = Babel.us_char
6525              local d
6526
6527              if crep and crep.data then
6528                item_base = data_nodes[crep.data]
6529              end
6530
6531              if crep then
6532                step = crep.step or 0
6533              end
6534
6535              if (not enabled) or (crep and next(crep) == nil) then -- = {}
6536                last_match = save_last     -- Optimization
6537                goto next
6538
6539              elseif crep == nil or crep.remove then
6540                node.remove(head, item)
6541                table.remove(w_nodes, sc)
6542                w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
6543                sc = sc - 1  -- Nothing has been inserted.
6544                last_match = utf8.offset(w, sc+1+step)
6545                goto next
6546
6547              elseif crep and crep.kashida then -- Experimental
6548                node.set_attribute(item,
6549                    Babel.attr_kashida,
6550                    crep.kashida)
6551                last_match = utf8.offset(w, sc+1+step)
```

```
6552              goto next
6553
6554        elseif crep and crep.string then
6555          local str = crep.string(matches)
6556          if str == '' then  -- Gather with nil
6557            node.remove(head, item)
6558            table.remove(w_nodes, sc)
6559            w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
6560            sc = sc - 1  -- Nothing has been inserted.
6561          else
6562            local loop_first = true
6563            for s in string.utfvalues(str) do
6564              d = node.copy(item_base)
6565              d.char = s
6566              if loop_first then
6567                loop_first = false
6568                head, new = node.insert_before(head, item, d)
6569                if sc == 1 then
6570                  word_head = head
6571                end
6572                w_nodes[sc] = d
6573                w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc+1)
6574              else
6575                sc = sc + 1
6576                head, new = node.insert_before(head, item, d)
6577                table.insert(w_nodes, sc, new)
6578                w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc)
6579              end
6580              if Babel.debug then
6581                print('.....', 'str')
6582                Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6583              end
6584            end  -- for
6585            node.remove(head, item)
6586          end  -- if ''
6587          last_match = utf8.offset(w, sc+1+step)
6588          goto next
6589
6590        elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
6591          d = node.new(7, 0)   -- (disc, discretionary)
6592          d.pre     = Babel.str_to_nodes(crep.pre, matches, item_base)
6593          d.post    = Babel.str_to_nodes(crep.post, matches, item_base)
6594          d.replace = Babel.str_to_nodes(crep.no, matches, item_base)
6595          d.attr = item_base.attr
6596          if crep.pre == nil then  -- TeXbook p96
6597            d.penalty = crep.penalty or tex.hyphenpenalty
6598          else
6599            d.penalty = crep.penalty or tex.exhyphenpenalty
6600          end
6601          placeholder = '|'
6602          head, new = node.insert_before(head, item, d)
6603
6604        elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
6605          -- ERROR
6606
6607        elseif crep and crep.penalty then
6608          d = node.new(14, 0)   -- (penalty, userpenalty)
6609          d.attr = item_base.attr
6610          d.penalty = crep.penalty
6611          head, new = node.insert_before(head, item, d)
6612
6613        elseif crep and crep.space then
6614          -- 655360 = 10 pt = 10 * 65536 sp
```

```
6615            d = node.new(12, 13)       -- (glue, spaceskip)
6616            local quad = font.getfont(item_base.font).size or 655360
6617            node.setglue(d, crep.space[1] * quad,
6618                            crep.space[2] * quad,
6619                            crep.space[3] * quad)
6620            if mode == 0 then
6621              placeholder = ' '
6622            end
6623            head, new = node.insert_before(head, item, d)
6624
6625          elseif crep and crep.spacefactor then
6626            d = node.new(12, 13)       -- (glue, spaceskip)
6627            local base_font = font.getfont(item_base.font)
6628            node.setglue(d,
6629              crep.spacefactor[1] * base_font.parameters['space'],
6630              crep.spacefactor[2] * base_font.parameters['space_stretch'],
6631              crep.spacefactor[3] * base_font.parameters['space_shrink'])
6632            if mode == 0 then
6633              placeholder = ' '
6634            end
6635            head, new = node.insert_before(head, item, d)
6636
6637          elseif mode == 0 and crep and crep.space then
6638            -- ERROR
6639
6640          end  -- ie replacement cases
6641
6642          -- Shared by disc, space and penalty.
6643          if sc == 1 then
6644            word_head = head
6645          end
6646          if crep.insert then
6647            w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc)
6648            table.insert(w_nodes, sc, new)
6649            last = last + 1
6650          else
6651            w_nodes[sc] = d
6652            node.remove(head, item)
6653            w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1)
6654          end
6655
6656          last_match = utf8.offset(w, sc+1+step)
6657
6658          ::next::
6659
6660        end  -- for each replacement
6661
6662        if Babel.debug then
6663            print('.....', '/')
6664            Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6665        end
6666
6667      end  -- for match
6668
6669    end  -- for patterns
6670
6671    ::next::
6672    word_head = nw
6673  end  -- for substring
6674  return head
6675 end
6676
6677 -- This table stores capture maps, numbered consecutively
```

```
6678 Babel.capture_maps = {}
6679
6680 -- The following functions belong to the next macro
6681 function Babel.capture_func(key, cap)
6682   local ret = "[[" .. cap:gsub('{([0-9])}', "]]..m[%1]..[[") .. "]]"
6683   local cnt
6684   local u = unicode.utf8
6685   ret, cnt = ret:gsub('{([0-9])|([^|]+)|(.-)}', Babel.capture_func_map)
6686   if cnt == 0 then
6687     ret = u.gsub(ret, '{(%x%x%x%x+)}',
6688           function (n)
6689             return u.char(tonumber(n, 16))
6690           end)
6691   end
6692   ret = ret:gsub("%[%[%]%]%.%.", '')
6693   ret = ret:gsub("%.%.%[%[%]%]", '')
6694   return key .. [[=function(m) return ]] .. ret .. [[ end]]
6695 end
6696
6697 function Babel.capt_map(from, mapno)
6698   return Babel.capture_maps[mapno][from] or from
6699 end
6700
6701 -- Handle the {n|abc|ABC} syntax in captures
6702 function Babel.capture_func_map(capno, from, to)
6703   local u = unicode.utf8
6704   from = u.gsub(from, '{(%x%x%x%x+)}',
6705         function (n)
6706           return u.char(tonumber(n, 16))
6707         end)
6708   to = u.gsub(to, '{(%x%x%x%x+)}',
6709         function (n)
6710           return u.char(tonumber(n, 16))
6711         end)
6712   local froms = {}
6713   for s in string.utfcharacters(from) do
6714     table.insert(froms, s)
6715   end
6716   local cnt = 1
6717   table.insert(Babel.capture_maps, {})
6718   local mlen = table.getn(Babel.capture_maps)
6719   for s in string.utfcharacters(to) do
6720     Babel.capture_maps[mlen][froms[cnt]] = s
6721     cnt = cnt + 1
6722   end
6723   return "]]..Babel.capt_map(m[" .. capno .. "]," ..
6724         (mlen) .. ")..".. "[["
6725 end
6726
6727 -- Create/Extend reversed sorted list of kashida weights:
6728 function Babel.capture_kashida(key, wt)
6729   wt = tonumber(wt)
6730   if Babel.kashida_wts then
6731     for p, q in ipairs(Babel.kashida_wts) do
6732       if wt  == q then
6733         break
6734       elseif wt > q then
6735         table.insert(Babel.kashida_wts, p, wt)
6736         break
6737       elseif table.getn(Babel.kashida_wts) == p then
6738         table.insert(Babel.kashida_wts, wt)
6739       end
6740     end
```

```
6741   else
6742     Babel.kashida_wts = { wt }
6743   end
6744   return 'kashida = ' .. wt
6745 end
6746 ⟨/transforms⟩
```

12.12   Lua: Auto bidi with `basic` and `basic-r`

The file babel-data-bidi.lua currently only contains data. It is a large and boring file and it is not shown here (see the generated file), but here is a sample:

```
[0x25]={d='et'},
[0x26]={d='on'},
[0x27]={d='on'},
[0x28]={d='on', m=0x29},
[0x29]={d='on', m=0x28},
[0x2A]={d='on'},
[0x2B]={d='es'},
[0x2C]={d='cs'},
```

For the meaning of these codes, see the Unicode standard.

Now the `basic-r` bidi mode. One of the aims is to implement a fast and simple bidi algorithm, with a single loop. I managed to do it for R texts, with a second smaller loop for a special case. The code is still somewhat chaotic, but its behavior is essentially correct. I cannot resist copying the following text from Emacs `bidi.c` (which also attempts to implement the bidi algorithm with a single loop):

> Arrrgh!! The UAX#9 algorithm is too deeply entrenched in the assumption of batch-style processing [...]. May the fleas of a thousand camels infest the armpits of those who design supposedly general-purpose algorithms by looking at their own implementations, and fail to consider other possible implementations!

Well, it took me some time to guess what the batch rules in UAX#9 actually mean (in other word, *what* they do and *why*, and not only *how*), but I think (or I hope) I've managed to understand them. In some sense, there are two bidi modes, one for numbers, and the other for text. Furthermore, setting just the direction in R text is not enough, because there are actually *two* R modes (set explicitly in Unicode with RLM and ALM). In babel the dir is set by a higher protocol based on the language/script, which in turn sets the correct dir (<l>, <r> or <al>).

From UAX#9: "Where available, markup should be used instead of the explicit formatting characters". So, this simple version just ignores formatting characters. Actually, most of that annex is devoted to how to handle them.

BD14-BD16 are not implemented. Unicode (and the W3C) are making a great effort to deal with some special problematic cases in "streamed" plain text. I don't think this is the way to go – particular issues should be fixed by a high level interface taking into account the needs of the document. And here is where luatex excels, because everything related to bidi writing is under our control.

```
6747 ⟨*basic-r⟩
6748 Babel = Babel or {}
6749
6750 Babel.bidi_enabled = true
6751
6752 require('babel-data-bidi.lua')
6753
6754 local characters = Babel.characters
6755 local ranges = Babel.ranges
6756
6757 local DIR = node.id("dir")
6758
6759 local function dir_mark(head, from, to, outer)
6760   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
6761   local d = node.new(DIR)
6762   d.dir = '+' .. dir
6763   node.insert_before(head, from, d)
6764   d = node.new(DIR)
```

```
6765    d.dir = '-' .. dir
6766    node.insert_after(head, to, d)
6767  end
6768
6769  function Babel.bidi(head, ispar)
6770    local first_n, last_n          -- first and last char with nums
6771    local last_es                  -- an auxiliary 'last' used with nums
6772    local first_d, last_d          -- first and last char in L/R block
6773    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):

```
6774    local strong = ('TRT' == tex.pardir) and 'r' or 'l'
6775    local strong_lr = (strong == 'l') and 'l' or 'r'
6776    local outer = strong
6777
6778    local new_dir = false
6779    local first_dir = false
6780    local inmath = false
6781
6782    local last_lr
6783
6784    local type_n = ''
6785
6786    for item in node.traverse(head) do
6787
6788      -- three cases: glyph, dir, otherwise
6789      if item.id == node.id'glyph'
6790        or (item.id == 7 and item.subtype == 2) then
6791
6792        local itemchar
6793        if item.id == 7 and item.subtype == 2 then
6794          itemchar = item.replace.char
6795        else
6796          itemchar = item.char
6797        end
6798        local chardata = characters[itemchar]
6799        dir = chardata and chardata.d or nil
6800        if not dir then
6801          for nn, et in ipairs(ranges) do
6802            if itemchar < et[1] then
6803              break
6804            elseif itemchar <= et[2] then
6805              dir = et[3]
6806              break
6807            end
6808          end
6809        end
6810        dir = dir or 'l'
6811        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).

```
6812        if new_dir then
6813          attr_dir = 0
6814          for at in node.traverse(item.attr) do
6815            if at.number == Babel.attr_dir then
6816              attr_dir = at.value % 3
6817            end
6818          end
```

```
6819        if attr_dir == 1 then
6820          strong = 'r'
6821        elseif attr_dir == 2 then
6822          strong = 'al'
6823        else
6824          strong = 'l'
6825        end
6826        strong_lr = (strong == 'l') and 'l' or 'r'
6827        outer = strong_lr
6828        new_dir = false
6829      end
6830
6831      if dir == 'nsm' then dir = strong end              -- W1
```

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

```
6832      dir_real = dir              -- We need dir_real to set strong below
6833      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:

```
6834      if strong == 'al' then
6835        if dir == 'en' then dir = 'an' end              -- W2
6836        if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
6837        strong_lr = 'r'                                -- W3
6838      end
```

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

```
6839    elseif item.id == node.id'dir' and not inmath then
6840      new_dir = true
6841      dir = nil
6842    elseif item.id == node.id'math' then
6843      inmath = (item.subtype == 0)
6844    else
6845      dir = nil          -- Not a char
6846    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>.

```
6847    if dir == 'en' or dir == 'an' or dir == 'et' then
6848      if dir ~= 'et' then
6849        type_n = dir
6850      end
6851      first_n = first_n or item
6852      last_n = last_es or item
6853      last_es = nil
6854    elseif dir == 'es' and last_n then -- W3+W6
6855      last_es = item
6856    elseif dir == 'cs' then            -- it's right - do nothing
6857    elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
6858      if strong_lr == 'r' and type_n ~= '' then
6859        dir_mark(head, first_n, last_n, 'r')
6860      elseif strong_lr == 'l' and first_d and type_n == 'an' then
6861        dir_mark(head, first_n, last_n, 'r')
6862        dir_mark(head, first_d, last_d, outer)
6863        first_d, last_d = nil, nil
6864      elseif strong_lr == 'l' and type_n ~= '' then
6865        last_d = last_n
6866      end
6867      type_n = ''
6868      first_n, last_n = nil, nil
6869    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:

```
6870    if dir == 'l' or dir == 'r' then
6871      if dir ~= outer then
6872        first_d = first_d or item
6873        last_d = item
6874      elseif first_d and dir ~= strong_lr then
6875        dir_mark(head, first_d, last_d, outer)
6876        first_d, last_d = nil, nil
6877      end
6878    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.

```
6879    if dir and not last_lr and dir ~= 'l' and outer == 'r' then
6880      item.char = characters[item.char] and
6881                  characters[item.char].m or item.char
6882    elseif (dir or new_dir) and last_lr ~= item then
6883      local mir = outer .. strong_lr .. (dir or outer)
6884      if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
6885        for ch in node.traverse(node.next(last_lr)) do
6886          if ch == item then break end
6887          if ch.id == node.id'glyph' and characters[ch.char] then
6888            ch.char = characters[ch.char].m or ch.char
6889          end
6890        end
6891      end
6892    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).

```
6893    if dir == 'l' or dir == 'r' then
6894      last_lr = item
6895      strong = dir_real            -- Don't search back - best save now
6896      strong_lr = (strong == 'l') and 'l' or 'r'
6897    elseif new_dir then
6898      last_lr = nil
6899    end
6900  end
```

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

```
6901  if last_lr and outer == 'r' then
6902    for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
6903      if characters[ch.char] then
6904        ch.char = characters[ch.char].m or ch.char
6905      end
6906    end
6907  end
6908  if first_n then
6909    dir_mark(head, first_n, last_n, outer)
6910  end
6911  if first_d then
6912    dir_mark(head, first_d, last_d, outer)
6913  end
```

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

```
6914  return node.prev(head) or head
```

```
6915 end
6916 ⟨/basic-r⟩
```

And here the Lua code for bidi=basic:

```
6917 ⟨*basic⟩
6918 Babel = Babel or {}
6919
6920 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
6921
6922 Babel.fontmap = Babel.fontmap or {}
6923 Babel.fontmap[0] = {}       -- l
6924 Babel.fontmap[1] = {}       -- r
6925 Babel.fontmap[2] = {}       -- al/an
6926
6927 Babel.bidi_enabled = true
6928 Babel.mirroring_enabled = true
6929
6930 require('babel-data-bidi.lua')
6931
6932 local characters = Babel.characters
6933 local ranges = Babel.ranges
6934
6935 local DIR = node.id('dir')
6936 local GLYPH = node.id('glyph')
6937
6938 local function insert_implicit(head, state, outer)
6939   local new_state = state
6940   if state.sim and state.eim and state.sim ~= state.eim then
6941     dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
6942     local d = node.new(DIR)
6943     d.dir = '+' .. dir
6944     node.insert_before(head, state.sim, d)
6945     local d = node.new(DIR)
6946     d.dir = '-' .. dir
6947     node.insert_after(head, state.eim, d)
6948   end
6949   new_state.sim, new_state.eim = nil, nil
6950   return head, new_state
6951 end
6952
6953 local function insert_numeric(head, state)
6954   local new
6955   local new_state = state
6956   if state.san and state.ean and state.san ~= state.ean then
6957     local d = node.new(DIR)
6958     d.dir = '+TLT'
6959     _, new = node.insert_before(head, state.san, d)
6960     if state.san == state.sim then state.sim = new end
6961     local d = node.new(DIR)
6962     d.dir = '-TLT'
6963     _, new = node.insert_after(head, state.ean, d)
6964     if state.ean == state.eim then state.eim = new end
6965   end
6966   new_state.san, new_state.ean = nil, nil
6967   return head, new_state
6968 end
6969
6970 -- TODO - \hbox with an explicit dir can lead to wrong results
6971 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
6972 -- was s made to improve the situation, but the problem is the 3-dir
6973 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
6974 -- well.
6975
```

```
6976 function Babel.bidi(head, ispar, hdir)
6977   local d    -- d is used mainly for computations in a loop
6978   local prev_d = ''
6979   local new_d = false
6980
6981   local nodes = {}
6982   local outer_first = nil
6983   local inmath = false
6984
6985   local glue_d = nil
6986   local glue_i = nil
6987
6988   local has_en = false
6989   local first_et = nil
6990
6991   local ATDIR = Babel.attr_dir
6992
6993   local save_outer
6994   local temp = node.get_attribute(head, ATDIR)
6995   if temp then
6996     temp = temp % 3
6997     save_outer = (temp == 0 and 'l') or
6998                  (temp == 1 and 'r') or
6999                  (temp == 2 and 'al')
7000   elseif ispar then              -- Or error? Shouldn't happen
7001     save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
7002   else                          -- Or error? Shouldn't happen
7003     save_outer = ('TRT' == hdir) and 'r' or 'l'
7004   end
7005     -- when the callback is called, we are just _after_ the box,
7006     -- and the textdir is that of the surrounding text
7007   -- if not ispar and hdir ~= tex.textdir then
7008   --   save_outer = ('TRT' == hdir) and 'r' or 'l'
7009   -- end
7010   local outer = save_outer
7011   local last = outer
7012   -- 'al' is only taken into account in the first, current loop
7013   if save_outer == 'al' then save_outer = 'r' end
7014
7015   local fontmap = Babel.fontmap
7016
7017   for item in node.traverse(head) do
7018
7019     -- In what follows, #node is the last (previous) node, because the
7020     -- current one is not added until we start processing the neutrals.
7021
7022     -- three cases: glyph, dir, otherwise
7023     if item.id == GLYPH
7024        or (item.id == 7 and item.subtype == 2) then
7025
7026       local d_font = nil
7027       local item_r
7028       if item.id == 7 and item.subtype == 2 then
7029         item_r = item.replace    -- automatic discs have just 1 glyph
7030       else
7031         item_r = item
7032       end
7033       local chardata = characters[item_r.char]
7034       d = chardata and chardata.d or nil
7035       if not d or d == 'nsm' then
7036         for nn, et in ipairs(ranges) do
7037           if item_r.char < et[1] then
7038             break
```

```
7039        elseif item_r.char <= et[2] then
7040          if not d then d = et[3]
7041          elseif d == 'nsm' then d_font = et[3]
7042          end
7043          break
7044        end
7045      end
7046    end
7047    d = d or 'l'
7048
7049    -- A short 'pause' in bidi for mapfont
7050    d_font = d_font or d
7051    d_font = (d_font == 'l' and 0) or
7052             (d_font == 'nsm' and 0) or
7053             (d_font == 'r' and 1) or
7054             (d_font == 'al' and 2) or
7055             (d_font == 'an' and 2) or nil
7056    if d_font and fontmap and fontmap[d_font][item_r.font] then
7057      item_r.font = fontmap[d_font][item_r.font]
7058    end
7059
7060    if new_d then
7061      table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
7062      if inmath then
7063        attr_d = 0
7064      else
7065        attr_d = node.get_attribute(item, ATDIR)
7066        attr_d = attr_d % 3
7067      end
7068      if attr_d == 1 then
7069        outer_first = 'r'
7070        last = 'r'
7071      elseif attr_d == 2 then
7072        outer_first = 'r'
7073        last = 'al'
7074      else
7075        outer_first = 'l'
7076        last = 'l'
7077      end
7078      outer = last
7079      has_en = false
7080      first_et = nil
7081      new_d = false
7082    end
7083
7084    if glue_d then
7085      if (d == 'l' and 'l' or 'r') ~= glue_d then
7086        table.insert(nodes, {glue_i, 'on', nil})
7087      end
7088      glue_d = nil
7089      glue_i = nil
7090    end
7091
7092  elseif item.id == DIR then
7093    d = nil
7094    if head ~= item then new_d = true end
7095
7096  elseif item.id == node.id'glue' and item.subtype == 13 then
7097    glue_d = d
7098    glue_i = item
7099    d = nil
7100
7101  elseif item.id == node.id'math' then
```

```
7102        inmath = (item.subtype == 0)
7103
7104    else
7105      d = nil
7106    end
7107
7108    -- AL <= EN/ET/ES      -- W2 + W3 + W6
7109    if last == 'al' and d == 'en' then
7110      d = 'an'           -- W3
7111    elseif last == 'al' and (d == 'et' or d == 'es') then
7112      d = 'on'           -- W6
7113    end
7114
7115    -- EN + CS/ES + EN     -- W4
7116    if d == 'en' and #nodes >= 2 then
7117      if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
7118          and nodes[#nodes-1][2] == 'en' then
7119        nodes[#nodes][2] = 'en'
7120      end
7121    end
7122
7123    -- AN + CS + AN        -- W4 too, because uax9 mixes both cases
7124    if d == 'an' and #nodes >= 2 then
7125      if (nodes[#nodes][2] == 'cs')
7126          and nodes[#nodes-1][2] == 'an' then
7127        nodes[#nodes][2] = 'an'
7128      end
7129    end
7130
7131    -- ET/EN               -- W5 + W7->l / W6->on
7132    if d == 'et' then
7133      first_et = first_et or (#nodes + 1)
7134    elseif d == 'en' then
7135      has_en = true
7136      first_et = first_et or (#nodes + 1)
7137    elseif first_et then       -- d may be nil here !
7138      if has_en then
7139        if last == 'l' then
7140          temp = 'l'     -- W7
7141        else
7142          temp = 'en'    -- W5
7143        end
7144      else
7145        temp = 'on'      -- W6
7146      end
7147      for e = first_et, #nodes do
7148        if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
7149      end
7150      first_et = nil
7151      has_en = false
7152    end
7153
7154    -- Force mathdir in math if ON (currently works as expected only
7155    -- with 'l')
7156    if inmath and d == 'on' then
7157      d = ('TRT' == tex.mathdir) and 'r' or 'l'
7158    end
7159
7160    if d then
7161      if d == 'al' then
7162        d = 'r'
7163        last = 'al'
7164      elseif d == 'l' or d == 'r' then
```

```
7165          last = d
7166        end
7167        prev_d = d
7168        table.insert(nodes, {item, d, outer_first})
7169      end
7170
7171      outer_first = nil
7172
7173   end
7174
7175   -- TODO -- repeated here in case EN/ET is the last node. Find a
7176   -- better way of doing things:
7177   if first_et then        -- dir may be nil here !
7178     if has_en then
7179       if last == 'l' then
7180         temp = 'l'      -- W7
7181       else
7182         temp = 'en'     -- W5
7183       end
7184     else
7185       temp = 'on'       -- W6
7186     end
7187     for e = first_et, #nodes do
7188       if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
7189     end
7190   end
7191
7192   -- dummy node, to close things
7193   table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
7194
7195   --------------- NEUTRAL -----------------
7196
7197   outer = save_outer
7198   last = outer
7199
7200   local first_on = nil
7201
7202   for q = 1, #nodes do
7203     local item
7204
7205     local outer_first = nodes[q][3]
7206     outer = outer_first or outer
7207     last = outer_first or last
7208
7209     local d = nodes[q][2]
7210     if d == 'an' or d == 'en' then d = 'r' end
7211     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
7212
7213     if d == 'on' then
7214       first_on = first_on or q
7215     elseif first_on then
7216       if last == d then
7217         temp = d
7218       else
7219         temp = outer
7220       end
7221       for r = first_on, q - 1 do
7222         nodes[r][2] = temp
7223         item = nodes[r][1]    -- MIRRORING
7224         if Babel.mirroring_enabled and item.id == GLYPH
7225             and temp == 'r' and characters[item.char] then
7226           local font_mode = ''
7227           if font.fonts[item.font].properties then
```

```lua
7228              font_mode = font.fonts[item.font].properties.mode
7229            end
7230          if font_mode ~= 'harf' and font_mode ~= 'plug' then
7231            item.char = characters[item.char].m or item.char
7232          end
7233        end
7234      end
7235      first_on = nil
7236    end
7237
7238    if d == 'r' or d == 'l' then last = d end
7239  end
7240
7241  --------------  IMPLICIT, REORDER ----------------
7242
7243  outer = save_outer
7244  last = outer
7245
7246  local state = {}
7247  state.has_r = false
7248
7249  for q = 1, #nodes do
7250
7251    local item = nodes[q][1]
7252
7253    outer = nodes[q][3] or outer
7254
7255    local d = nodes[q][2]
7256
7257    if d == 'nsm' then d = last end              -- W1
7258    if d == 'en' then d = 'an' end
7259    local isdir = (d == 'r' or d == 'l')
7260
7261    if outer == 'l' and d == 'an' then
7262      state.san = state.san or item
7263      state.ean = item
7264    elseif state.san then
7265      head, state = insert_numeric(head, state)
7266    end
7267
7268    if outer == 'l' then
7269      if d == 'an' or d == 'r' then      -- im -> implicit
7270        if d == 'r' then state.has_r = true end
7271        state.sim = state.sim or item
7272        state.eim = item
7273      elseif d == 'l' and state.sim and state.has_r then
7274        head, state = insert_implicit(head, state, outer)
7275      elseif d == 'l' then
7276        state.sim, state.eim, state.has_r = nil, nil, false
7277      end
7278    else
7279      if d == 'an' or d == 'l' then
7280        if nodes[q][3] then -- nil except after an explicit dir
7281          state.sim = item  -- so we move sim 'inside' the group
7282        else
7283          state.sim = state.sim or item
7284        end
7285        state.eim = item
7286      elseif d == 'r' and state.sim then
7287        head, state = insert_implicit(head, state, outer)
7288      elseif d == 'r' then
7289        state.sim, state.eim = nil, nil
7290      end
```

```
7291    end
7292
7293    if isdir then
7294      last = d              -- Don't search back - best save now
7295    elseif d == 'on' and state.san  then
7296      state.san = state.san or item
7297      state.ean = item
7298    end
7299
7300  end
7301
7302  return node.prev(head) or head
7303 end
7304 ⟨/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.

```
7305 ⟨*nil⟩
7306 \ProvidesLanguage{nil}[⟨⟨date⟩⟩ ⟨⟨version⟩⟩ Nil language]
7307 \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.

```
7308 \ifx\l@nil\@undefined
7309   \newlanguage\l@nil
7310   \@namedef{bbl@hyphendata@\the\l@nil}{{}{}}%  Remove warning
7311   \let\bbl@elt\relax
7312   \edef\bbl@languages{%  Add it to the list of languages
7313     \bbl@languages\bbl@elt{nil}{\the\l@nil}{}{}}
7314 \fi
```

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

```
7315 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}
```

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

\captionnil
  \datenil
```
7316 \let\captionsnil\@empty
7317 \let\datenil\@empty
```

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

```
7318 \def\bbl@inidata@nil{%
7319   \bbl@elt{identification}{tag.ini}{und}%
```

```
7320    \bbl@elt{identification}{load.level}{0}%
7321    \bbl@elt{identification}{charset}{utf8}%
7322    \bbl@elt{identification}{version}{1.0}%
7323    \bbl@elt{identification}{date}{2022-05-16}%
7324    \bbl@elt{identification}{name.local}{nil}%
7325    \bbl@elt{identification}{name.english}{nil}%
7326    \bbl@elt{identification}{name.babel}{nil}%
7327    \bbl@elt{identification}{tag.bcp47}{und}%
7328    \bbl@elt{identification}{language.tag.bcp47}{und}%
7329    \bbl@elt{identification}{tag.opentype}{dflt}%
7330    \bbl@elt{identification}{script.name}{Latin}%
7331    \bbl@elt{identification}{script.tag.bcp47}{Latn}%
7332    \bbl@elt{identification}{script.tag.opentype}{DFLT}%
7333    \bbl@elt{identification}{level}{1}%
7334    \bbl@elt{identification}{encodings}{}%
7335    \bbl@elt{identification}{derivate}{no}}
7336 \@namedef{bbl@tbcp@nil}{und}
7337 \@namedef{bbl@lbcp@nil}{und}
7338 \@namedef{bbl@lotf@nil}{dflt}
7339 \@namedef{bbl@elname@nil}{nil}
7340 \@namedef{bbl@lname@nil}{nil}
7341 \@namedef{bbl@esname@nil}{Latin}
7342 \@namedef{bbl@sname@nil}{Latin}
7343 \@namedef{bbl@sbcp@nil}{Latn}
7344 \@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.

```
7345 \ldf@finish{nil}
7346 ⟨/nil⟩
```

15 Calendars

The code for specific calendars are placed in the specific files, loaded when requested by an ini file in the `identification` section with `require.calendars`.

15.1 Islamic

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

```
7347 ⟨*ca-islamic⟩
7348 \ExplSyntaxOn
7349 \def\bbl@fpmod#1#2{(#1-#2*floor(#1/#2))}
7350 \def\bbl@cs@gregleap#1{%
7351   (\bbl@fpmod{#1}{4} == 0) &&
7352     (!((\bbl@fpmod{#1}{100} == 0) && (\bbl@fpmod{#1}{400} != 0)))}
7353 \def\bbl@ca@jd#1#2#3{% year, month, day
7354   \fp_eval:n{ 1721424.5   + (365 * (#1 - 1)) +
7355     floor((#1 - 1) / 4)   + (-floor((#1 - 1) / 100)) +
7356     floor((#1 - 1) / 400) + floor((((367 * #2) - 362) / 12) +
7357     ((#2 <= 2) ? 0 : (\bbl@cs@gregleap{#1} ? -1 : -2)) + #3) }}
7358 % == islamic (default)
7359 % Not yet implemented
7360 \def\bbl@ca@islamic#1-#2-#3\@@#4#5#6{}
```

The Civil calendar.

```
7361 \def\bbl@cs@isltojd#1#2#3{ % year, month, day
7362   ((#3 + ceil(29.5 * (#2 - 1)) +
7363   (#1 - 1) * 354 + floor((3 + (11 * #1)) / 30) +
7364   1948439.5) - 1) }
7365 \@namedef{bbl@ca@islamic-civil++}{\bbl@ca@islamicvl@x{+2}}
7366 \@namedef{bbl@ca@islamic-civil+}{\bbl@ca@islamicvl@x{+1}}
```

```
7367 \@namedef{bbl@ca@islamic-civil}{\bbl@ca@islamicvl@x{}}
7368 \@namedef{bbl@ca@islamic-civil-}{\bbl@ca@islamicvl@x{-1}}
7369 \@namedef{bbl@ca@islamic-civil--}{\bbl@ca@islamicvl@x{-2}}
7370 \def\bbl@ca@islamicvl@x#1#2-#3-#4\@@#5#6#7{%
7371   \edef\bbl@tempa{%
7372     \fp_eval:n{ floor(\bbl@ca@jd{#2}{#3}{#4})+0.5 #1}}%
7373   \edef#5{%
7374     \fp_eval:n{ floor(((30*(\bbl@tempa-1948439.5)) + 10646)/10631) }}%
7375   \edef#6{\fp_eval:n{
7376     min(12,ceil((\bbl@tempa-(29+\bbl@cs@isltojd{#5}{1}{1}))/29.5)+1) }}%
7377   \edef#7{\fp_eval:n{ \bbl@tempa - \bbl@cs@isltojd{#5}{#6}{1} + 1} }}
```

The Umm al-Qura calendar, used mainly in Saudi Arabia. Based on islamdate_today.js on
https://webspace.science.uu.nl/. Copyright 1999-2002 by Ray Stott. Free to use if credited.
Since the main aim is to provide a suitable \today, and maybe some close dates, data just covers
Hijri ∼1435/∼1460 (Gregorian ∼2014/∼2038).

```
7378 \def\bbl@ca@umalqura@data{56660, 56690,56719,56749,56778,56808,%
7379   56837,56867,56897,56926,56956,56985,57015,57044,57074,57103,%
7380   57133,57162,57192,57221,57251,57280,57310,57340,57369,57399,%
7381   57429,57458,57487,57517,57546,57576,57605,57634,57664,57694,%
7382   57723,57753,57783,57813,57842,57871,57901,57930,57959,57989,%
7383   58018,58048,58077,58107,58137,58167,58196,58226,58255,58285,%
7384   58314,58343,58373,58402,58432,58461,58491,58521,58551,58580,%
7385   58610,58639,58669,58698,58727,58757,58786,58816,58845,58875,%
7386   58905,58934,58964,58994,59023,59053,59082,59111,59141,59170,%
7387   59200,59229,59259,59288,59318,59348,59377,59407,59436,59466,%
7388   59495,59525,59554,59584,59613,59643,59672,59702,59731,59761,%
7389   59791,59820,59850,59879,59909,59939,59968,59997,60027,60056,%
7390   60086,60115,60145,60174,60204,60234,60264,60293,60323,60352,%
7391   60381,60411,60440,60469,60499,60528,60558,60588,60618,60648,%
7392   60677,60707,60736,60765,60795,60824,60853,60883,60912,60942,%
7393   60972,61002,61031,61061,61090,61120,61149,61179,61208,61237,%
7394   61267,61296,61326,61356,61385,61415,61445,61474,61504,61533,%
7395   61563,61592,61621,61651,61680,61710,61739,61769,61799,61828,%
7396   61858,61888,61917,61947,61976,62006,62035,62064,62094,62123,%
7397   62153,62182,62212,62242,62271,62301,62331,62360,62390,62419,%
7398   62448,62478,62507,62537,62566,62596,62625,62655,62685,62715,%
7399   62744,62774,62803,62832,62862,62891,62921,62950,62980,63009,%
7400   63039,63069,63099,63128,63157,63187,63216,63246,63275,63305,%
7401   63334,63363,63393,63423,63453,63482,63512,63541,63571,63600,%
7402   63630,63659,63689,63718,63747,63777,63807,63836,63866,63895,%
7403   63925,63955,63984,64014,64043,64073,64102,64131,64161,64190,%
7404   64220,64249,64279,64309,64339,64368,64398,64427,64457,64486,%
7405   64515,64545,64574,64603,64633,64663,64692,64722,64752,64782,%
7406   64811,64841,64870,64899,64929,64958,64987,65017,65047,65076,%
7407   65106,65136,65166,65195,65225,65254,65283,65313,65342,65371,%
7408   65401,65431,65460,65490,65520}
7409 \@namedef{bbl@ca@islamic-umalqura+}{\bbl@ca@islamcuqr@x{+1}}
7410 \@namedef{bbl@ca@islamic-umalqura}{\bbl@ca@islamcuqr@x{}}
7411 \@namedef{bbl@ca@islamic-umalqura-}{\bbl@ca@islamcuqr@x{-1}}
7412 \def\bbl@ca@islamcuqr@x#1#2-#3-#4\@@#5#6#7{%
7413   \edef\bbl@tempd{\fp_eval:n{ % (Julian) day
7414     \bbl@ca@jd{#2}{#3}{#4} + 0.5 - 2400000 #1}}%
7415   \count@\@ne
7416   \bbl@foreach\bbl@ca@umalqura@data{%
7417     \advance\count@\@ne
7418     \ifnum##1>\bbl@tempd\else
7419       \edef\bbl@tempe{\the\count@}%
7420       \edef\bbl@tempb{##1}%
7421     \fi}%
7422   \edef\bbl@templ{\fp_eval:n{ \bbl@tempe + 16260 + 949 }}% month~lunar
7423   \edef\bbl@tempa{\fp_eval:n{ floor((\bbl@templ - 1 ) / 12) }}% annus
7424   \edef#5{\fp_eval:n{ \bbl@tempa + 1  }}%
```

```
7425    \edef#6{\fp_eval:n{ \bbl@templ - (12 * \bbl@tempa) }}%
7426    \edef#7{\fp_eval:n{ \bbl@tempd - \bbl@tempb + 1 }}}}
7427 \ExplSyntaxOff
7428 ⟨/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.

```
7429 ⟨*ca-hebrew⟩
7430 \newcount\bbl@cntcommon
7431 \def\bbl@remainder#1#2#3{%
7432   #3 = #1                  %  c = a
7433   \divide #3 by #2         %  c = a/b
7434   \multiply #3 by -#2      %  c = -b(a/b)
7435   \advance #3 by #1 }%       %  c = a - b(a/b)
7436 \newif\ifbbl@divisible
7437 \def\bbl@checkifdivisible#1#2{%
7438   {\countdef\tmp = 0 % \tmp == \count0 - temporary variable
7439   \bbl@remainder{#1}{#2}{\tmp}%
7440   \ifnum \tmp = 0
7441       \global\bbl@divisibletrue
7442   \else
7443       \global\bbl@divisiblefalse
7444   \fi}}
7445 \newif\ifbbl@gregleap
7446 \def\bbl@ifgregleap#1{%
7447   \bbl@checkifdivisible{#1}{4}%
7448   \ifbbl@divisible
7449       \bbl@checkifdivisible{#1}{100}%
7450       \ifbbl@divisible
7451           \bbl@checkifdivisible{#1}{400}%
7452           \ifbbl@divisible
7453               \bbl@gregleaptrue
7454           \else
7455               \bbl@gregleapfalse
7456           \fi
7457       \else
7458           \bbl@gregleaptrue
7459       \fi
7460   \else
7461       \bbl@gregleapfalse
7462   \fi
7463   \ifbbl@gregleap}
7464 \def\bbl@gregdayspriormonths#1#2#3{% no month number 0
7465     {#3 = \ifcase #1 0 \or 0 \or 31 \or 59 \or 90 \or 120 \or 151 \or
7466          181 \or 212 \or 243 \or 273 \or 304 \or 334 \fi
7467     \bbl@ifgregleap{#2}%
7468         \ifnum #1 > 2        % if month after February
7469             \advance #3 by 1  % add leap day
7470         \fi
7471     \fi
7472     \global\bbl@cntcommon = #3}%
7473     #3 = \bbl@cntcommon}
7474 \def\bbl@gregdaysprioryears#1#2{%
7475   {\countdef\tmpc = 4      % \tmpc==\count4
7476   \countdef\tmpb = 2      % \tmpb==\count2
7477   \tmpb = #1              %
7478   \advance \tmpb by -1    %
7479   \tmpc = \tmpb           % \tmpc = \tmpb = year-1
7480   \multiply \tmpc by 365 % Days in prior years =
```

```
7481    #2 = \tmpc              % = 365*(year-1) ...
7482    \tmpc = \tmpb           %
7483    \divide \tmpc by 4      % \tmpc = (year-1)/4
7484    \advance #2 by \tmpc    % ... plus Julian leap days ...
7485    \tmpc = \tmpb           %
7486    \divide \tmpc by 100    % \tmpc = (year-1)/100
7487    \advance #2 by -\tmpc   % ... minus century years ...
7488    \tmpc = \tmpb           %
7489    \divide \tmpc by 400    % \tmpc = (year-1)/400
7490    \advance #2 by \tmpc    % ... plus 4-century years.
7491    \global\bbl@cntcommon = #2}%
7492    #2 = \bbl@cntcommon}
7493 \def\bbl@absfromgreg#1#2#3#4{%
7494    {\countdef\tmpd = 0      % \tmpd==\count0
7495    #4 = #1                  % days so far this month
7496    \bbl@gregdayspriormonths{#2}{#3}{\tmpd}%
7497    \advance #4 by \tmpd     % add days in prior months
7498    \bbl@gregdaysprioryears{#3}{\tmpd}%
7499    \advance #4 by \tmpd     % add days in prior years
7500    \global\bbl@cntcommon = #4}%
7501    #4 = \bbl@cntcommon}
7502 \newif\ifbbl@hebrleap
7503 \def\bbl@checkleaphebryear#1{%
7504    {\countdef\tmpa = 0      % \tmpa==\count0
7505    \countdef\tmpb = 1       % \tmpb==\count1
7506    \tmpa = #1
7507    \multiply \tmpa by 7
7508    \advance \tmpa by 1
7509    \bbl@remainder{\tmpa}{19}{\tmpb}%
7510    \ifnum \tmpb < 7         % \tmpb = (7*year+1)%19
7511        \global\bbl@hebrleaptrue
7512    \else
7513        \global\bbl@hebrleapfalse
7514    \fi}}
7515 \def\bbl@hebrelapsedmonths#1#2{%
7516    {\countdef\tmpa = 0      % \tmpa==\count0
7517    \countdef\tmpb = 1       % \tmpb==\count1
7518    \countdef\tmpc = 2       % \tmpc==\count2
7519    \tmpa = #1               %
7520    \advance \tmpa by -1     %
7521    #2 = \tmpa               % #2 = \tmpa = year-1
7522    \divide #2 by 19         % Number of complete Meton cycles
7523    \multiply #2 by 235      % #2 = 235*((year-1)/19)
7524    \bbl@remainder{\tmpa}{19}{\tmpb}% \tmpa = years%19-years this cycle
7525    \tmpc = \tmpb            %
7526    \multiply \tmpb by 12    %
7527    \advance #2 by \tmpb     % add regular months this cycle
7528    \multiply \tmpc by 7     %
7529    \advance \tmpc by 1      %
7530    \divide \tmpc by 19      % \tmpc = (1+7*((year-1)%19))/19 -
7531    \advance #2 by \tmpc     %  add leap months
7532    \global\bbl@cntcommon = #2}%
7533    #2 = \bbl@cntcommon}
7534 \def\bbl@hebrelapseddays#1#2{%
7535    {\countdef\tmpa = 0      % \tmpa==\count0
7536    \countdef\tmpb = 1       % \tmpb==\count1
7537    \countdef\tmpc = 2       % \tmpc==\count2
7538    \bbl@hebrelapsedmonths{#1}{#2}%
7539    \tmpa = #2               %
7540    \multiply \tmpa by 13753 %
7541    \advance \tmpa by 5604   % \tmpa=MonthsElapsed*13758 + 5604
7542    \bbl@remainder{\tmpa}{25920}{\tmpc}% \tmpc == ConjunctionParts
7543    \divide \tmpa by 25920
```

```
7544    \multiply #2 by 29
7545    \advance #2 by 1
7546    \advance #2 by \tmpa      %  #2 = 1 + MonthsElapsed*29 +
7547    \bbl@remainder{#2}{7}{\tmpa}% %  \tmpa == DayOfWeek
7548    \ifnum \tmpc < 19440
7549        \ifnum \tmpc < 9924
7550        \else                  % New moon at 9 h. 204 p. or later
7551            \ifnum \tmpa = 2  % on Tuesday ...
7552                \bbl@checkleaphebryear{#1}% of a common year
7553                \ifbbl@hebrleap
7554                \else
7555                    \advance #2 by 1
7556                \fi
7557            \fi
7558        \fi
7559        \ifnum \tmpc < 16789
7560        \else                    % New moon at 15 h. 589 p. or later
7561            \ifnum \tmpa = 1   % on Monday ...
7562                \advance #1 by -1
7563                \bbl@checkleaphebryear{#1}% at the end of leap year
7564                \ifbbl@hebrleap
7565                    \advance #2 by 1
7566                \fi
7567            \fi
7568        \fi
7569    \else
7570        \advance #2 by 1         %  new moon at or after midday
7571    \fi
7572    \bbl@remainder{#2}{7}{\tmpa}%  %  \tmpa == DayOfWeek
7573    \ifnum \tmpa = 0           %  if Sunday ...
7574        \advance #2 by 1
7575    \else                      %
7576        \ifnum \tmpa = 3       %  Wednesday ...
7577            \advance #2 by 1
7578        \else
7579            \ifnum \tmpa = 5   %  or Friday
7580                \advance #2 by 1
7581            \fi
7582        \fi
7583    \fi
7584    \global\bbl@cntcommon = #2}%
7585  #2 = \bbl@cntcommon}
7586 \def\bbl@daysinhebryear#1#2{%
7587  {\countdef\tmpe = 12    % \tmpe==\count12
7588   \bbl@hebrelapseddays{#1}{\tmpe}%
7589   \advance #1 by 1
7590   \bbl@hebrelapseddays{#1}{#2}%
7591   \advance #2 by -\tmpe
7592   \global\bbl@cntcommon = #2}%
7593  #2 = \bbl@cntcommon}
7594 \def\bbl@hebrdayspriormonths#1#2#3{%
7595  {\countdef\tmpf= 14    % \tmpf==\count14
7596   #3 = \ifcase #1        % Days in prior month of regular year
7597        0 \or          % no month number 0
7598        0 \or          % Tishri
7599       30 \or          % Heshvan
7600       59 \or          % Kislev
7601       89 \or          % Tebeth
7602      118 \or          % Shebat
7603      148 \or          % Adar I
7604      148 \or          % Adar II
7605      177 \or          % Nisan
7606      207 \or          % Iyar
```

```
7607        236 \or          % Sivan
7608        266 \or          % Tammuz
7609        295 \or          % Av
7610        325 \or          % Elul
7611        400              % Dummy
7612    \fi
7613    \bbl@checkleaphebryear{#2}%
7614    \ifbbl@hebrleap            % in leap year
7615        \ifnum #1 > 6          % if month after Adar I
7616            \advance #3 by 30  % add  30 days
7617        \fi
7618    \fi
7619    \bbl@daysinhebryear{#2}{\tmpf}%
7620    \ifnum #1 > 3
7621        \ifnum \tmpf = 353    %
7622            \advance #3 by -1  %
7623        \fi                    %   Short Kislev
7624        \ifnum \tmpf = 383    %
7625            \advance #3 by -1  %
7626        \fi                    %
7627    \fi
7628    \ifnum #1 > 2
7629        \ifnum \tmpf = 355    %
7630            \advance #3 by 1   %
7631        \fi                    %   Long Heshvan
7632        \ifnum \tmpf = 385    %
7633            \advance #3 by 1   %
7634        \fi                    %
7635    \fi
7636    \global\bbl@cntcommon = #3}%
7637    #3 = \bbl@cntcommon}
7638 \def\bbl@absfromhebr#1#2#3#4{%
7639    {#4 = #1
7640    \bbl@hebrdayspriormonths{#2}{#3}{#1}%
7641    \advance #4 by #1         % Add days in prior months this year
7642    \bbl@hebrelapseddays{#3}{#1}%
7643    \advance #4 by #1         % Add days in prior years
7644    \advance #4 by -1373429    % Subtract days before Gregorian
7645    \global\bbl@cntcommon = #4}%    %   01.01.0001
7646    #4 = \bbl@cntcommon}
7647 \def\bbl@hebrfromgreg#1#2#3#4#5#6{%
7648    {\countdef\tmpx= 17        % \tmpx==\count17
7649    \countdef\tmpy= 18        % \tmpy==\count18
7650    \countdef\tmpz= 19        % \tmpz==\count19
7651    #6 = #3                   %
7652    \global\advance #6 by 3761 % approximation from above
7653    \bbl@absfromgreg{#1}{#2}{#3}{#4}%
7654    \tmpz = 1   \tmpy = 1
7655    \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
7656    \ifnum \tmpx > #4              %
7657        \global\advance #6 by -1 % Hyear = Gyear + 3760
7658        \bbl@absfromhebr{\tmpz}{\tmpy}{#6}{\tmpx}%
7659    \fi                           %
7660    \advance #4 by -\tmpx     % Days in this year
7661    \advance #4 by 1          %
7662    #5 = #4                   %
7663    \divide #5 by 30          % Approximation for month from below
7664    \loop                     % Search for month
7665        \bbl@hebrdayspriormonths{#5}{#6}{\tmpx}%
7666        \ifnum \tmpx < #4
7667            \advance #5 by 1
7668            \tmpy = \tmpx
7669    \repeat
```

209

```
7670     \global\advance #5 by -1
7671     \global\advance #4 by -\tmpy}}
7672 \newcount\bbl@hebrday \newcount\bbl@hebrmonth \newcount\bbl@hebryear
7673 \newcount\bbl@gregday \newcount\bbl@gregmonth \newcount\bbl@gregyear
7674 %
7675 \def\bbl@ca@hebrew#1-#2-#3\@@#4#5#6{%
7676     \bbl@gregday=#3 \bbl@gregmonth=#2 \bbl@gregyear=#1
7677     \bbl@hebrfromgreg
7678       {\bbl@gregday}{\bbl@gregmonth}{\bbl@gregyear}%
7679       {\bbl@hebrday}{\bbl@hebrmonth}{\bbl@hebryear}%
7680     \edef#4{\the\bbl@hebryear}%
7681     \edef#5{\the\bbl@hebrmonth}%
7682     \edef#6{\the\bbl@hebrday}}
7683 ⟨/ca-hebrew⟩
```

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

17.1   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`.

```
7684 ⟨∗bplain | blplain⟩
7685 \catcode`\{=1 % left brace is begin-group character
7686 \catcode`\}=2 % right brace is end-group character
7687 \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).

```
7688 \openin 0 hyphen.cfg
7689 \ifeof0
7690 \else
7691   \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.

```
7692   \def\input #1 {%
7693     \let\input\a
7694     \a hyphen.cfg
7695     \let\a\undefined
7696   }
7697 \fi
7698 ⟨/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.

```
7699 ⟨bplain⟩\a plain.tex
7700 ⟨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.

```
7701 ⟨bplain⟩\def\fmtname{babel-plain}
7702 ⟨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.

17.2   Emulating some LaTeX features

The file babel.def expects some definitions made in the LaTeX2ε 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).

```
7703 ⟨⟨*Emulate LaTeX⟩⟩ ≡
7704 \def\@empty{}
7705 \def\loadlocalcfg#1{%
7706   \openin0#1.cfg
7707   \ifeof0
7708     \closein0
7709   \else
7710     \closein0
7711     {\immediate\write16{********************************}%
7712      \immediate\write16{* Local config file #1.cfg used}%
7713      \immediate\write16{*}%
7714      }
7715     \input #1.cfg\relax
7716   \fi
7717   \@endofldf}
```

17.3   General tools

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

```
7718 \long\def\@firstofone#1{#1}
7719 \long\def\@firstoftwo#1#2{#1}
7720 \long\def\@secondoftwo#1#2{#2}
7721 \def\@nnil{\@nil}
7722 \def\@gobbletwo#1#2{}
7723 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
7724 \def\@star@or@long#1{%
7725   \@ifstar
7726   {\let\l@ngrel@x\relax#1}%
7727   {\let\l@ngrel@x\long#1}}
7728 \let\l@ngrel@x\relax
7729 \def\@car#1#2\@nil{#1}
7730 \def\@cdr#1#2\@nil{#2}
7731 \let\@typeset@protect\relax
7732 \let\protected@edef\edef
7733 \long\def\@gobble#1{}
7734 \edef\@backslashchar{\expandafter\@gobble\string\\}
7735 \def\strip@prefix#1>{}
7736 \def\g@addto@macro#1#2{{%
7737     \toks@\expandafter{#1#2}%
7738     \xdef#1{\the\toks@}}}
7739 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
7740 \def\@nameuse#1{\csname #1\endcsname}
7741 \def\@ifundefined#1{%
7742   \expandafter\ifx\csname#1\endcsname\relax
7743     \expandafter\@firstoftwo
7744   \else
7745     \expandafter\@secondoftwo
```

```
7746   \fi}
7747 \def\@expandtwoargs#1#2#3{%
7748   \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
7749 \def\zap@space#1 #2{%
7750   #1%
7751   \ifx#2\@empty\else\expandafter\zap@space\fi
7752   #2}
7753 \let\bbl@trace\@gobble
7754 \def\bbl@error#1#2{%
7755   \begingroup
7756     \newlinechar=`\^^J
7757     \def\\{^^J(babel) }%
7758     \errhelp{#2}\errmessage{\\#1}%
7759   \endgroup}
7760 \def\bbl@warning#1{%
7761   \begingroup
7762     \newlinechar=`\^^J
7763     \def\\{^^J(babel) }%
7764     \message{\\#1}%
7765   \endgroup}
7766 \let\bbl@infowarn\bbl@warning
7767 \def\bbl@info#1{%
7768   \begingroup
7769     \newlinechar=`\^^J
7770     \def\\{^^J}%
7771     \wlog{#1}%
7772   \endgroup}
```

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

```
7773 \ifx\@preamblecmds\@undefined
7774   \def\@preamblecmds{}
7775 \fi
7776 \def\@onlypreamble#1{%
7777   \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
7778     \@preamblecmds\do#1}}
7779 \@onlypreamble\@onlypreamble
```

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

```
7780 \def\begindocument{%
7781   \@begindocumenthook
7782   \global\let\@begindocumenthook\@undefined
7783   \def\do##1{\global\let##1\@undefined}%
7784   \@preamblecmds
7785   \global\let\do\noexpand}
7786 \ifx\@begindocumenthook\@undefined
7787   \def\@begindocumenthook{}
7788 \fi
7789 \@onlypreamble\@begindocumenthook
7790 \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.

```
7791 \def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
7792 \@onlypreamble\AtEndOfPackage
7793 \def\@endofldf{}
7794 \@onlypreamble\@endofldf
7795 \let\bbl@afterlang\@empty
7796 \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.

```
7797 \catcode`\&=\z@
7798 \ifx&if@filesw\@undefined
7799   \expandafter\let\csname if@filesw\expandafter\endcsname
7800     \csname iffalse\endcsname
7801 \fi
7802 \catcode`\&=4
```

Mimick LaTeX's commands to define control sequences.

```
7803 \def\newcommand{\@star@or@long\new@command}
7804 \def\new@command#1{%
7805   \@testopt{\@newcommand#1}0}
7806 \def\@newcommand#1[#2]{%
7807   \@ifnextchar [{\@xargdef#1[#2]}%
7808                 {\@argdef#1[#2]}}
7809 \long\def\@argdef#1[#2]#3{%
7810   \@yargdef#1\@ne{#2}{#3}}
7811 \long\def\@xargdef#1[#2][#3]#4{%
7812   \expandafter\def\expandafter#1\expandafter{%
7813     \expandafter\@protected@testopt\expandafter #1%
7814     \csname\string#1\expandafter\endcsname{#3}}%
7815   \expandafter\@yargdef \csname\string#1\endcsname
7816   \tw@{#2}{#4}}
7817 \long\def\@yargdef#1#2#3{%
7818   \@tempcnta#3\relax
7819   \advance \@tempcnta \@ne
7820   \let\@hash@\relax
7821   \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
7822   \@tempcntb #2%
7823   \@whilenum\@tempcntb <\@tempcnta
7824   \do{%
7825     \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
7826     \advance\@tempcntb \@ne}%
7827   \let\@hash@##%
7828   \l@ngrel@x\expandafter\def\expandafter#1\reserved@a}
7829 \def\providecommand{\@star@or@long\provide@command}
7830 \def\provide@command#1{%
7831   \begingroup
7832     \escapechar\m@ne\xdef\@gtempa{{\string#1}}%
7833   \endgroup
7834   \expandafter\@ifundefined\@gtempa
7835     {\def\reserved@a{\new@command#1}}%
7836     {\let\reserved@a\relax
7837      \def\reserved@a{\new@command\reserved@a}}%
7838   \reserved@a}%
7839 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
7840 \def\declare@robustcommand#1{%
7841   \edef\reserved@a{\string#1}%
7842   \def\reserved@b{#1}%
7843   \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
7844   \edef#1{%
7845     \ifx\reserved@a\reserved@b
7846       \noexpand\x@protect
7847       \noexpand#1%
7848     \fi
7849     \noexpand\protect
7850     \expandafter\noexpand\csname
7851       \expandafter\@gobble\string#1 \endcsname
7852   }%
7853   \expandafter\new@command\csname
7854     \expandafter\@gobble\string#1 \endcsname
7855 }
7856 \def\x@protect#1{%
7857   \ifx\protect\@typeset@protect\else
```

```
7858        \@x@protect#1%
7859      \fi
7860 }
7861 \catcode`\&=\z@  % Trick to hide conditionals
7862    \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.

```
7863    \def\bbl@tempa{\csname newif\endcsname&ifin@}
7864 \catcode`\&=4
7865 \ifx\in@\@undefined
7866    \def\in@#1#2{%
7867      \def\in@@##1#1##2##3\in@@{%
7868        \ifx\in@##2\in@false\else\in@true\fi}%
7869      \in@@#2#1\in@\in@@}
7870 \else
7871    \let\bbl@tempa\@empty
7872 \fi
7873 \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).

```
7874 \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.

```
7875 \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 2$_\varepsilon$ versions; just enough to make things work in plain TeXenvironments.

```
7876 \ifx\@tempcnta\@undefined
7877    \csname newcount\endcsname\@tempcnta\relax
7878 \fi
7879 \ifx\@tempcntb\@undefined
7880    \csname newcount\endcsname\@tempcntb\relax
7881 \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).

```
7882 \ifx\bye\@undefined
7883    \advance\count10 by -2\relax
7884 \fi
7885 \ifx\@ifnextchar\@undefined
7886    \def\@ifnextchar#1#2#3{%
7887      \let\reserved@d=#1%
7888      \def\reserved@a{#2}\def\reserved@b{#3}%
7889      \futurelet\@let@token\@ifnch}
7890    \def\@ifnch{%
7891      \ifx\@let@token\@sptoken
7892        \let\reserved@c\@xifnch
7893      \else
7894        \ifx\@let@token\reserved@d
7895          \let\reserved@c\reserved@a
7896        \else
7897          \let\reserved@c\reserved@b
7898        \fi
7899      \fi
7900      \reserved@c}
7901    \def\:{\let\@sptoken= } \:  % this makes \@sptoken a space token
```

214

```
7902    \def\:{\@xifnch} \expandafter\def\: {\futurelet\@let@token\@ifnch}
7903 \fi
7904 \def\@testopt#1#2{%
7905    \@ifnextchar[{#1}{#1[#2]}}
7906 \def\@protected@testopt#1{%
7907    \ifx\protect\@typeset@protect
7908      \expandafter\@testopt
7909    \else
7910      \@x@protect#1%
7911    \fi}
7912 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
7913        #2\relax}\fi}
7914 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
7915        \else\expandafter\@gobble\fi{#1}}
```

17.4   Encoding related macros

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

```
7916 \def\DeclareTextCommand{%
7917    \@dec@text@cmd\providecommand
7918 }
7919 \def\ProvideTextCommand{%
7920    \@dec@text@cmd\providecommand
7921 }
7922 \def\DeclareTextSymbol#1#2#3{%
7923    \@dec@text@cmd\chardef#1{#2}#3\relax
7924 }
7925 \def\@dec@text@cmd#1#2#3{%
7926    \expandafter\def\expandafter#2%
7927        \expandafter{%
7928          \csname#3-cmd\expandafter\endcsname
7929          \expandafter#2%
7930          \csname#3\string#2\endcsname
7931        }%
7932 %   \let\@ifdefinable\@rc@ifdefinable
7933    \expandafter#1\csname#3\string#2\endcsname
7934 }
7935 \def\@current@cmd#1{%
7936    \ifx\protect\@typeset@protect\else
7937      \noexpand#1\expandafter\@gobble
7938    \fi
7939 }
7940 \def\@changed@cmd#1#2{%
7941    \ifx\protect\@typeset@protect
7942        \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
7943          \expandafter\ifx\csname ?\string#1\endcsname\relax
7944            \expandafter\def\csname ?\string#1\endcsname{%
7945              \@changed@x@err{#1}%
7946            }%
7947          \fi
7948          \global\expandafter\let
7949            \csname\cf@encoding \string#1\expandafter\endcsname
7950            \csname ?\string#1\endcsname
7951        \fi
7952        \csname\cf@encoding\string#1%
7953          \expandafter\endcsname
7954    \else
7955      \noexpand#1%
7956    \fi
7957 }
7958 \def\@changed@x@err#1{%
7959    \errhelp{Your command will be ignored, type <return> to proceed}%
7960    \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
```

```
7961 \def\DeclareTextCommandDefault#1{%
7962     \DeclareTextCommand#1?%
7963 }
7964 \def\ProvideTextCommandDefault#1{%
7965     \ProvideTextCommand#1?%
7966 }
7967 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
7968 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
7969 \def\DeclareTextAccent#1#2#3{%
7970     \DeclareTextCommand#1{#2}[1]{\accent#3 ##1}
7971 }
7972 \def\DeclareTextCompositeCommand#1#2#3#4{%
7973     \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
7974     \edef\reserved@b{\string##1}%
7975     \edef\reserved@c{%
7976         \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
7977     \ifx\reserved@b\reserved@c
7978         \expandafter\expandafter\expandafter\ifx
7979             \expandafter\@car\reserved@a\relax\relax\@nil
7980             \@text@composite
7981         \else
7982             \edef\reserved@b##1{%
7983                 \def\expandafter\noexpand
7984                     \csname#2\string#1\endcsname####1{%
7985                     \noexpand\@text@composite
7986                         \expandafter\noexpand\csname#2\string#1\endcsname
7987                         ####1\noexpand\@empty\noexpand\@text@composite
7988                         {##1}%
7989                 }%
7990             }%
7991             \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
7992         \fi
7993         \expandafter\def\csname\expandafter\string\csname
7994             #2\endcsname\string#1-\string#3\endcsname{#4}
7995     \else
7996         \errhelp{Your command will be ignored, type <return> to proceed}%
7997         \errmessage{\string\DeclareTextCompositeCommand\space used on
7998             inappropriate command \protect#1}
7999     \fi
8000 }
8001 \def\@text@composite#1#2#3\@text@composite{%
8002     \expandafter\@text@composite@x
8003         \csname\string#1-\string#2\endcsname
8004 }
8005 \def\@text@composite@x#1#2{%
8006     \ifx#1\relax
8007         #2%
8008     \else
8009         #1%
8010     \fi
8011 }
8012 %
8013 \def\@strip@args#1:#2-#3\@strip@args{#2}
8014 \def\DeclareTextComposite#1#2#3#4{%
8015     \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
8016     \bgroup
8017         \lccode`\@=#4%
8018         \lowercase{%
8019     \egroup
8020         \reserved@a @%
8021     }%
8022 }
8023 %
```

```
8024 \def\UseTextSymbol#1#2{#2}
8025 \def\UseTextAccent#1#2#3{}
8026 \def\@use@text@encoding#1{}
8027 \def\DeclareTextSymbolDefault#1#2{%
8028    \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
8029 }
8030 \def\DeclareTextAccentDefault#1#2{%
8031    \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
8032 }
8033 \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.

```
8034 \DeclareTextAccent{\"}{OT1}{127}
8035 \DeclareTextAccent{\'}{OT1}{19}
8036 \DeclareTextAccent{\^}{OT1}{94}
8037 \DeclareTextAccent{\`}{OT1}{18}
8038 \DeclareTextAccent{\~}{OT1}{126}
```

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

```
8039 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
8040 \DeclareTextSymbol{\textquotedblright}{OT1}{`\"}
8041 \DeclareTextSymbol{\textquoteleft}{OT1}{`\`}
8042 \DeclareTextSymbol{\textquoteright}{OT1}{`\'}
8043 \DeclareTextSymbol{\i}{OT1}{16}
8044 \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.

```
8045 \ifx\scriptsize\@undefined
8046    \let\scriptsize\sevenrm
8047 \fi
```

And a few more "dummy" definitions.

```
8048 \def\languagename{english}%
8049 \let\bbl@opt@shorthands\@nnil
8050 \def\bbl@ifshorthand#1#2#3{#2}%
8051 \let\bbl@language@opts\@empty
8052 \ifx\babeloptionstrings\@undefined
8053    \let\bbl@opt@strings\@nnil
8054 \else
8055    \let\bbl@opt@strings\babeloptionstrings
8056 \fi
8057 \def\BabelStringsDefault{generic}
8058 \def\bbl@tempa{normal}
8059 \ifx\babeloptionmath\bbl@tempa
8060    \def\bbl@mathnormal{\noexpand\textormath}
8061 \fi
8062 \def\AfterBabelLanguage#1#2{}
8063 \ifx\BabelModifiers\@undefined\let\BabelModifiers\relax\fi
8064 \let\bbl@afterlang\relax
8065 \def\bbl@opt@safe{BR}
8066 \ifx\@uclclist\@undefined\let\@uclclist\@empty\fi
8067 \ifx\bbl@trace\@undefined\def\bbl@trace#1{}\fi
8068 \expandafter\newif\csname ifbbl@single\endcsname
8069 \chardef\bbl@bidimode\z@
8070 ⟨/Emulate LaTeX⟩
```

A proxy file:

```
8071 ⟨*plain⟩
8072 \input babel.def
8073 ⟨/plain⟩
```

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

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

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

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