Babel

Johannes L. Braams
Original author

Javier Bezos
Current maintainer

Localization and internationalization

Unicode
T$_E$X
pdfT$_E$X
LuaT$_E$X
XeT$_E$X

Contents

Troubleshoooting

Part I

User guide

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

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

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

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

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

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

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

1 The user interface

1.1 Monolingual documents

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

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

```
\documentclass{article}

\usepackage[T1]{fontenc}

\usepackage[french]{babel}

\begin{document}

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

\end{document}
```

Now consider something like:

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

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

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

```
\documentclass[russian]{article}

\usepackage{babel}

\babelfont{rm}{DejaVu Serif}

\begin{document}

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

\end{document}
```

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

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

Or the more explanatory:

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

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

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 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:

LUATEX/XETEX
```
\documentclass[english]{article}
\usepackage{babel}

\babelfont[russian]{rm}{FreeSerif}

\begin{document}

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

\end{document}
```

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

---

[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.

```
! 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.

1.6 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.

---

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

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 \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.

\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).

<dl>
<dt>\babelensure</dt>
<dd>

`[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.
</dd>
</dl>

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:

---

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

```
    ! 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*.)

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:

---

[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.

```
\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 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.

---

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

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]

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.

---

[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.

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}
```

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 200 of these files containing the basic data required for a locale.

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 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, but still problematic).

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

```
\babelprovide[import, hyphenrules=+]{lao}
\babelpatterns[lao]{1ກ 1ຍ 1ຂ 1ງ 1ກ 1ຖ} % 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 seems 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]	bg	Bulgarian[ul]
agq	Aghem	bm	Bambara
ak	Akan	bn	Bangla[ul]
am	Amharic[ul]	bo	Tibetan[u]
ar	Arabic[ul]	brx	Bodo
ar-DZ	Arabic[ul]	bs-Cyrl	Bosnian
ar-MA	Arabic[ul]	bs-Latn	Bosnian[ul]
ar-SY	Arabic[ul]	bs	Bosnian[ul]
as	Assamese	ca	Catalan[ul]
asa	Asu	ce	Chechen
ast	Asturian[ul]	cgg	Chiga
az-Cyrl	Azerbaijani	chr	Cherokee
az-Latn	Azerbaijani	ckb	Central Kurdish
az	Azerbaijani[ul]	cop	Coptic
bas	Basaa	cs	Czech[ul]
be	Belarusian[ul]	cu	Church Slavic
bem	Bemba	cu-Cyrs	Church Slavic
bez	Bena	cu-Glag	Church Slavic

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

Code	Language	Code	Language
mr	Marathi[ul]	shi	Tachelhit
ms-BN	Malay[l]	si	Sinhala
ms-SG	Malay[l]	sk	Slovak[ul]
ms	Malay[ul]	sl	Slovenian[ul]
mt	Maltese	smn	Inari Sami
mua	Mundang	sn	Shona
my	Burmese	so	Somali
mzn	Mazanderani	sq	Albanian[ul]
naq	Nama	sr-Cyrl-BA	Serbian[ul]
nb	Norwegian Bokmål[ul]	sr-Cyrl-ME	Serbian[ul]
nd	North Ndebele	sr-Cyrl-XK	Serbian[ul]
ne	Nepali	sr-Cyrl	Serbian[ul]
nl	Dutch[ul]	sr-Latn-BA	Serbian[ul]
nmg	Kwasio	sr-Latn-ME	Serbian[ul]
nn	Norwegian Nynorsk[ul]	sr-Latn-XK	Serbian[ul]
nnh	Ngiemboon	sr-Latn	Serbian[ul]
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 Tamazight
se	Northern Sami[ul]	zh-Hans-HK	Chinese
seh	Sena	zh-Hans-MO	Chinese
ses	Koyraboro Senni	zh-Hans-SG	Chinese
sg	Sango	zh-Hans	Chinese
shi-Latn	Tachelhit	zh-Hant-HK	Chinese
shi-Tfng	Tachelhit		

| zh-Hant-MO | Chinese | zh | Chinese |
| zh-Hant | Chinese | 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

---

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

english-gb
english-newzealand
english-nz
english-unitedkingdom
english-unitedstates
english-us
english
esperanto
estonian
ewe
ewondo
faroese
filipino
finnish
french-be
french-belgium
french-ca
french-canada
french-ch
french-lu
french-luxembourg
french-switzerland
french
friulian
fulah
galician
ganda
georgian
german-at
german-austria
german-ch
german-switzerland
german
greek
gujarati
gusii
hausa-gh
hausa-ghana
hausa-ne
hausa-niger
hausa
hawaiian
hebrew
hindi
hungarian
icelandic
igbo
inarisami
indonesian
interlingua
irish
italian
japanese
jolafonyi

kabuverdianu
kabyle
kako
kalaallisut
kalenjin
kamba
kannada
kashmiri
kazakh
khmer
kikuyu
kinyarwanda
konkani
korean
koyraborosenni
koyrachiini
kwasio
kyrgyz
lakota
langi
lao
latvian
lingala
lithuanian
lowersorbian
lsorbian
lubakatanga
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	vai-latn
ukrainian	vai-vai
uppersorbian	vai-vaii
urdu	vai
usenglish	vietnam
usorbian	vietnamese
uyghur	vunjo
uzbek-arab	walser
uzbek-arabic	welsh
uzbek-cyrillic	westernfrisian
uzbek-cyrl	yangben
uzbek-latin	yiddish
uzbek-latn	yoruba
uzbek	zarma
vai-latin	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.

---

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

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

LUATEX/XETEX
```
\documentclass{article}

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

\babelprovide[import]{hebrew}

\babelfont{rm}{FreeSerif}

\begin{document}

Svenska \foreignlanguage{hebrew}{עִבְרִית} svenska.

\end{document}
```

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

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

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

EXAMPLE Here is how to do it:

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

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

NOTE You may load fontspec explicitly. For example:

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

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

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

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

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

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

TROUBLESHOOTING *Package 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:

30

```
\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 is mainly in Polytonic Greek, 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]{babel}
    \babelprovide[import, main]{polytonicgreek}
```

Remerber there is an alternative syntax for the latter:

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

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 scrips, like Thai, and
CJK.

```

intrapenalty= ⟨*penalty*⟩

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

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

Languages providing native digits in all or some variants are:

| Arabic | Central Kurdish | Khmer | Northern Luri | Nepali |
| Assamese | Dzongkha | Kannada | Malayalam | Odia |
| Bangla | Persian | Konkani | Marathi | Punjabi |
| Tibetar | Gujarati | Kashmiri | Burmese | Pashto |
| Bodo | Hindi | Lao | Mazanderani | Tamil |

| Telugu | Uyghur | Uzbek | Cantonese |
|--------|--------|-------|-----------|
| Thai   | Urdu   | Vai   | Chinese   |

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.

New 4.41   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`
Belarusan, Bulgarian, Macedonian, Serbian `lower, upper`
Bengali `alphabetic`
Coptic `epact, lower.letters`
Hebrew `letters` (neither geresh nor gershayim yet)
Hindi `alphabetic`
Armenian `lower.letter, upper.letter`
Japanese `hiragana, hiragana.iroha, katakana, katakana.iroha, circled.katakana, informal, formal, cjk-earthly-branch, cjk-heavenly-stem, fullwidth.lower.alpha, fullwidth.upper.alpha`
Georgian `letters`
Greek `lower.modern, upper.modern, lower.ancient, upper.ancient` (all with keraia)
Khmer `consonant`
Korean `consonant, syllabe, hanja.informal, hanja.formal, hangul.formal, cjk-earthly-branch, cjk-heavenly-stem, 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`

Chinese `cjk-earthly-branch`, `cjk-heavenly-stem`, `fullwidth.lower.alpha`, `fullwidth.upper.alpha`

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

> **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.

`\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.

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.

NOTE ini files are loaded with `\babelprovide` and also when languages are selected if there is a `\babelfont`. To ensure the ini files are loaded (and therefore the corresponding data) even if these two conditions are not met, write `\BabelEnsureInfo` in the preamble.

`\localeid`

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

NOTE 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 (where it makes sense) as an attribute, too.

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.

`\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). 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`.

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

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

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}`.

---

[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.

Czech, Polish, Slovak	`oneletter.nobreak`	Converts a space after a non-syllabic preposition or conjunction into a non-breaking space.
Greek	`diaeresis.hyphen`	Removes the diaeresis above iota and upsilon if hyphenated just before. It works with the three variants.
Greek	`transliteration.omega`	Although he provided combinations are not exactly the same, 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.
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.
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.
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.

`\babelposthyphenation` $\{\langle hyphenrules\text{-}name\rangle\}\{\langle lua\text{-}pattern\rangle\}\{\langle replacement\rangle\}$

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.

See the babel site for a more detailed description and some examples. It also describes a few additional replacement types (`string`, `penalty`).

Although the main purpose of this command is non-standard hyphenation, it may actually be used for other transformations (after hyphenation is applied, so you must take discretionaries into account).

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

`\babelprehyphenation` {⟨*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.

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.[16]

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.[17]

\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 is progress in the latter, too, but for example cases 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

---

[16]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.

[17]But still defined for backwards compatibility.

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

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

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

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

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

```
\documentclass{article}

\usepackage[bidi=basic]{babel}

\babelprovide[import, main]{arabic}

\babelfont{rm}{FreeSerif}

\begin{document}

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

\end{document}
```

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

```
\documentclass{book}

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

\babelprovide[onchar=ids fonts]{arabic}

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

\begin{document}

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

\end{document}
```

43

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.[18]

`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).

---

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

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

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

graphics modifies the picture environment so that the whole figure is L but the text is R. It *does not* work with the standard picture, and *pict2e* is required. It attempts to do the same for pgf/tikz. Somewhat experimental. New 3.32 .

extras is used for miscellaneous readjustments which do not fit into the previous groups. Currently redefines in luatex \underline and \LaTeX2e New 3.19 .

EXAMPLE Typically, in an Arabic document you would need:

```
\usepackage[bidi=basic,
            layout=counters.tabular]{babel}
```

\babelsublr {⟨*lr-text*⟩}

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

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

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

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

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

\BabelPatchSection {⟨*section-name*⟩}

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

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

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

New 3.17 Something like:

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

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

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

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

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

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

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

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

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

1.25 Language attributes

\languageattribute

This is a user-level command, to be used in the preamble of a document (after \usepackage[...]{babel}), that declares which attributes are to be used for a given language. It takes two arguments: the first is the name of the language; the second, a (list of) attribute(s) to be used. Attributes must be set in the preamble and only once – they cannot be turned on and off. The command checks whether the language is known in this document and whether the attribute(s) are known for this language.
Very often, using a *modifier* in a package option is better.
Several language definition files use their own methods to set options. For example, french uses \frenchsetup, magyar (1.5) uses \magyarOptions; modifiers provided by spanish have no attribute counterparts. Macros setting options are also used (eg, \ProsodicMarksOn in latin).

1.26 Hooks

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

\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.

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

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

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

Afrikaans afrikaans
Azerbaijani azerbaijani
Basque basque
Breton breton
Bulgarian bulgarian
Catalan catalan
Croatian croatian
Czech czech
Danish danish
Dutch dutch
English english, USenglish, american, UKenglish, british, canadian, australian, newzealand
Esperanto esperanto
Estonian estonian
Finnish finnish
French french, francais, canadien, acadian
Galician galician
German austrian, german, germanb, ngerman, naustrian
Greek greek, polutonikogreek
Hebrew hebrew
Icelandic icelandic
Indonesian indonesian (bahasa, indon, bahasai)
Interlingua interlingua
Irish Gaelic irish
Italian italian
Latin latin
Lower Sorbian lowersorbian
Malay malay, melayu (bahasam)
North Sami samin
Norwegian norsk, nynorsk
Polish polish
Portuguese portuguese, brazilian (portuges, brazil)[19]
Romanian romanian
Russian russian
Scottish Gaelic scottish
Spanish spanish
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.

---

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

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

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

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

1.28 Unicode character properties in luatex

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

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

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

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

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

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

1.29 Tweaking some features

\babeladjust   {⟨*key-value-list*⟩}

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

49

1.30 Tips, workarounds, known issues and notes

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

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

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

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

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

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

- For the hyphenation to work correctly, lccodes cannot change, because TeX only takes into account the values when the paragraph is hyphenated, i.e., when it has been finished.[20] So, if you write a chunk of French text with `\foreignlanguage`, the apostrophes might not be taken into account. This is a limitation of TeX, not of babel. Alternatively, you may use `\useshorthands` to activate `'` and `\defineshorthand`, or redefine `\textquoteright` (the latter is called by the non-ASCII right quote).

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

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

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

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.

---

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

babelbib Multilingual bibliographies.

microtype Adjusts the typesetting according to some languages (kerning and spacing). Ligatures can be disabled.

substitutefont Combines fonts in several encodings.

mkpattern Generates hyphenation patterns.

tracklang Tracks which languages have been requested.

ucharclasses (xetex) Switches fonts when you switch from one Unicode block to another.

zhspacing Spacing for CJK documents in xetex.

1.31 Current and future work

The current work is focused on the so-called complex scripts in luatex. In 8-bit engines, babel provided a basic support for bidi text as part of the style for Hebrew, but it is somewhat unsatisfactory and internally replaces some hardwired commands by other hardwired commands (generic changes would be much better).

Useful additions would be, for example, time, currency, addresses and personal names.[21]. But that is the easy part, because they don't require modifying the LaTeX internals.

Calendars (Arabic, Persian, Indic, etc.) are under study.

Also interesting are differences in the sentence structure or related to it. For example, in Basque the number precedes the name (including chapters), in Hungarian "from (1)" is "(1)-ből", but "from (3)" is "(3)-ból", in Spanish an item labelled "3.º" may be referred to as either "ítem 3.º" or "3.ᵉʳ ítem", and so on.

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

1.32 Tentative and experimental code

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

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 wiki 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).

---

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

New 3.9q   With luatex, however, patterns are loaded on the fly when requested by the language (except the "0th" language, typically english, which is preloaded always).[22] Until 3.9n, this task was delegated to the package luatex-hyphen, by Khaled Hosny, Élie Roux, and Manuel Pégourié-Gonnard, and required an extra file named `language.dat.lua`, but now a new mechanism has been devised based solely on `language.dat`. **You must rebuild the formats** if upgrading from a previous version. You may want to have a local `language.dat` for a particular project (for example, a book on Chemistry).[23]

2.1   Format

In that file the person who maintains a TeX environment has to record for which languages he has hyphenation patterns *and* in which files these are stored[24]. When hyphenation exceptions are stored in a separate file this can be indicated by naming that file *after* the file with the hyphenation patterns.

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

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

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

You may also set the font encoding the patterns are intended for by following the language name by a colon and the encoding code.[25] For example:

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

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

A typical error when using babel is the following:

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

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

---

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

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

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

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

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

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

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

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

- The language definition files must define five macros, used to activate and deactivate the language-specific definitions. These macros are `\⟨lang⟩hyphenmins`, `\captions⟨lang⟩`, `\date⟨lang⟩`, `\extras⟨lang⟩` and `\noextras⟨lang⟩`(the last two may be left empty); where ⟨lang⟩ is either the name of the language definition file or the name of the LaTeX option that is to be used. These macros and their functions are discussed below. You must define all or none for a language (or a dialect); defining, say, `\date⟨lang⟩` but not `\captions⟨lang⟩` does not raise an error but can lead to unexpected results.

- When a language definition file is loaded, it can define `\l@⟨lang⟩` to be a dialect of `\language0` when `\l@⟨lang⟩` is undefined.

- Language names must be all lowercase. If an unknown language is selected, babel will attempt setting it after lowercasing its name.

- The semantics of modifiers is not defined (on purpose). In most cases, they will just be simple separated options (eg, `spanish`), but a language might require, say, a set of options organized as a tree with suboptions (in such a case, the recommended separator is /).

Some recommendations:

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

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

- Avoid adding things to `\noextras⟨lang⟩` except for umlauthigh and friends, `\bbl@deactivate`, `\bbl@(non)frenchspacing`, and language-specific macros. Use always, if possible, `\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.[26]

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

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

3.1 Guidelines for contributed languages

Currently, the easiest way to contribute a new language is by taking one the the 500 or so `ini` templates available on GitHub as a basis. Just make a pull request o dowonload it and then, after filling the fields, sent it to me. Fell free to ask for help or to make feature requests.
As to `ldf` files, now language files are "outsourced" and are located in a separate directory (`/macros/latex/contrib/babel-contrib`), so that they are contributed directly to CTAN (please, do not send to me language styles just to upload them to CTAN).
Of course, placing your style files in this directory is not mandatory, but if you want to do it, here are a few guidelines.

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

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

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

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

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

3.2 Basic macros

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

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

`\adddialect` The macro `\adddialect` can be used when two languages can (or must) use the same hyphenation patterns. This can also be useful for languages for which no patterns are preloaded in the format. In such cases the default behavior of the babel system is to define

---

[26]But not removed, for backward compatibility.

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

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

	(Assigning `\lefthyphenmin` and `\righthyphenmin` directly in `\extras<lang>` has no effect.)
`\providehyphenmins`	The macro `\providehyphenmins` should be used in the language definition files to set `\lefthyphenmin` and `\righthyphenmin`. This macro will check whether these parameters were provided by the hyphenation file before it takes any action. If these values have been already set, this command is ignored (currently, default pattern files do *not* set them).
`\captions⟨lang⟩`	The macro `\captions⟨lang⟩` defines the macros that hold the texts to replace the original hard-wired texts.
`\date⟨lang⟩`	The macro `\date⟨lang⟩` defines `\today`.
`\extras⟨lang⟩`	The macro `\extras⟨lang⟩` contains all the extra definitions needed for a specific language. This macro, like the following, is a hook – you can add things to it, but it must not be used directly.
`\noextras⟨lang⟩`	Because we want to let the user switch between languages, but we do not know what state TeX might be in after the execution of `\extras⟨lang⟩`, a macro that brings TeX into a predefined state is needed. It will be no surprise that the name of this macro is `\noextras⟨lang⟩`.
`\bbl@declare@ttribute`	This is a command to be used in the language definition files for declaring a language attribute. It takes three arguments: the name of the language, the attribute to be defined, and the code to be executed when the attribute is to be used.
`\main@language`	To postpone the activation of the definitions needed for a language until the beginning of a document, all language definition files should use `\main@language` instead of `\selectlanguage`. This will just store the name of the language, and the proper language will be activated at the start of the document.
`\ProvidesLanguage`	The macro `\ProvidesLanguage` should be used to identify the language definition files. Its syntax is similar to the syntax of the LaTeX command `\ProvidesPackage`.
`\LdfInit`	The macro `\LdfInit` performs a couple of standard checks that must be made at the beginning of a language definition file, such as checking the category code of the @-sign, preventing the `.ldf` file from being processed twice, etc.
`\ldf@quit`	The macro `\ldf@quit` does work needed if a `.ldf` file was processed earlier. This includes resetting the category code of the @-sign, preparing the language to be activated at `\begin{document}` time, and ending the input stream.
`\ldf@finish`	The macro `\ldf@finish` does work needed at the end of each `.ldf` file. This includes resetting the category code of the @-sign, loading a local configuration file, and preparing the language to be activated at `\begin{document}` time.
`\loadlocalcfg`	After processing a language definition file, LaTeX can be instructed to load a local configuration file. This file can, for instance, be used to add strings to `\captions⟨lang⟩` to support local document classes. The user will be informed that this configuration file has been loaded. This macro is called by `\ldf@finish`.
`\substitutefontfamily`	(Deprecated.) This command takes three arguments, a font encoding and two font family names. It creates a font description file for the first font in the given encoding. This `.fd` file will instruct LaTeX to use a font from the second family when a font from the first family in the given encoding seems to be needed.

3.3 Skeleton

Here is the basic structure of an ldf file, with a language, a dialect and an attribute. Strings are best defined using the method explained in sec. 3.8 (babel 3.9 and later).

```
\ProvidesLanguage{<language>}
     [2016/04/23 v0.0 <Language> support from the babel system]
\LdfInit{<language>}{captions<language>}

\ifx\undefined\l@<language>
  \@nopatterns{<Language>}
  \adddialect\l@<language>0
\fi

\adddialect\l@<dialect>\l@<language>

\bbl@declare@ttribute{<language>}{<attrib>}{%
  \expandafter\addto\expandafter\extras<language>
  \expandafter{\extras<attrib><language>}%
  \let\captions<language>\captions<attrib><language>}

\providehyphenmins{<language>}{\tw@\thr@@}

\StartBabelCommands*{<language>}{captions}
\SetString\chaptername{<chapter name>}
% More strings

\StartBabelCommands*{<language>}{date}
\SetString\monthiname{<name of first month>}
% More strings

\StartBabelCommands*{<dialect>}{captions}
\SetString\chaptername{<chapter name>}
% More strings

\StartBabelCommands*{<dialect>}{date}
\SetString\monthiname{<name of first month>}
% More strings

\EndBabelCommands

\addto\extras<language>{}
\addto\noextras<language>{}
\let\extras<dialect>\extras<language>
\let\noextras<dialect>\noextras<language>

\ldf@finish{<language>}
```

NOTE If for some reason you want to load a package in your style, you should be aware it cannot be done directly in the ldf file, but it can be delayed with \AtEndOfPackage. Macros from external packages can be used *inside* definitions in the ldf itself (for example, \extras<language>), but if executed directly, the code must be placed inside \AtEndOfPackage. A trivial example illustrating these points is:

```
\AtEndOfPackage{%
  \RequirePackage{dingbat}%        Delay package
  \savebox{\myeye}{\eye}}%         And direct usage
```

```
\newsavebox{\myeye}
\newcommand\myanchor{\anchor}%    But OK inside command
```

3.4   Support for active characters

In quite a number of language definition files, active characters are introduced. To facilitate this, some support macros are provided.

\initiate@active@char   The internal macro \initiate@active@char is used in language definition files to instruct LaTeX to give a character the category code 'active'. When a character has been made active it will remain that way until the end of the document. Its definition may vary.

\bbl@activate   The command \bbl@activate is used to change the way an active character expands.
\bbl@deactivate   \bbl@activate 'switches on' the active behavior of the character. \bbl@deactivate lets the active character expand to its former (mostly) non-active self.

\declare@shorthand   The macro \declare@shorthand is used to define the various shorthands. It takes three arguments: the name for the collection of shorthands this definition belongs to; the character (sequence) that makes up the shorthand, i.e. ~ or "a; and the code to be executed when the shorthand is encountered. (It does *not* raise an error if the shorthand character has not been "initiated".)

\bbl@add@special   The TeXbook states: "Plain TeX includes a macro called \dospecials that is essentially a set
\bbl@remove@special   macro, representing the set of all characters that have a special category code." [4, p. 380] It is used to set text 'verbatim'. To make this work if more characters get a special category code, you have to add this character to the macro \dospecial. LaTeX adds another macro called \@sanitize representing the same character set, but without the curly braces. The macros \bbl@add@special⟨*char*⟩ and \bbl@remove@special⟨*char*⟩ add and remove the character ⟨*char*⟩ to these two sets.

3.5   Support for saving macro definitions

Language definition files may want to *re*define macros that already exist. Therefore a mechanism for saving (and restoring) the original definition of those macros is provided. We provide two macros for this[27].

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

\babel@savevariable   A second macro is provided to save the current value of a variable. In this context, anything that is allowed after the \the primitive is considered to be a variable. The macro takes one argument, the ⟨*variable*⟩.
The effect of the preceding macros is to append a piece of code to the current definition of \originalTeX. When \originalTeX is expanded, this code restores the previous definition of the control sequence or the previous value of the variable.

3.6   Support for extending macros

\addto   The macro \addto{⟨*control sequence*⟩}{⟨*TeX code*⟩} can be used to extend the definition of a macro. The macro need not be defined (ie, it can be undefined or \relax). This macro can, for instance, be used in adding instructions to a macro like \extrasenglish.
Be careful when using this macro, because depending on the case the assignment can be either global (usually) or local (sometimes). That does not seem very consistent, but this behavior is preserved for backward compatibility. If you are using etoolbox, by Philipp Lehman, consider using the tools provided by this package instead of \addto.

---

[27]This mechanism was introduced by Bernd Raichle.

3.7   Macros common to a number of languages

\bbl@allowhyphens   In several languages compound words are used. This means that when TeX has to hyphenate such a compound word, it only does so at the '-' that is used in such words. To allow hyphenation in the rest of such a compound word, the macro \bbl@allowhyphens can be used.

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

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

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

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

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

3.8   Encoding-dependent strings

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

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

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

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

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

The ⟨language-list⟩ specifies which languages the block is intended for. A block is taken into account only if the \CurrentOption is listed here. Alternatively, you can define \BabelLanguages to a comma-separated list of languages to be defined (if undefined, \StartBabelCommands sets it to \CurrentOption). You may write \CurrentOption as the language, but this is discouraged – a explicit name (or names) is much better and clearer. A "selector" is a name to be used as value in package option strings, optionally followed by extra info about the encodings to be used. The name unicode must be used for xetex and luatex (the key strings has also other two special values: generic and encoded). If a string is set several times (because several blocks are read), the first one takes precedence (ie, it works much like \providecommand). Encoding info is charset= followed by a charset, which if given sets how the strings should be translated to the internal representation used by the engine, typically utf8, which is the only value supported currently (default is no translations). Note charset is applied by luatex and xetex when reading the file, not when the macro or string is used in the document.

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

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

The ⟨*category*⟩ is either `captions`, `date` or `extras`. You must stick to these three categories, even if no error is raised when using other name.[28] It may be empty, too, but in such a case using `\SetString` is an error (but not `\SetCase`).

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

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

\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}
```

---

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

```
  \SetString\today{\number\day.~%
    \csname month\romannumeral\month name\endcsname\space
    \number\year}

\StartBabelCommands{german,austrian}{captions}
  \SetString\prefacename{Vorwort}
  [etc.]

\EndBabelCommands
```

When used in ldf files, previous values of \⟨*category*⟩⟨*language*⟩ are overridden, which means the old way to define strings still works and used by default (to be precise, is first set to undefined and then strings are added). However, when used in the preamble or in a package, new settings are added to the previous ones, if the language exists (in the babel sense, ie, if \date⟨*language*⟩ exists).

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

The starred version just forces strings to take a value – if not set as package option, then the default for the engine is used. This is not done by default to prevent backward incompatibilities, but if you are creating a new language this version is better. It's up to the maintainers of the current languages to decide if using it is appropriate.[29]

\EndBabelCommands   Marks the end of the series of blocks.

\AfterBabelCommands   {⟨*code*⟩}

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

\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

---

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

arguments take precedence over the optional one. This command, unlike \SetString, is executed always (even without strings), and it is intended for minor readjustments only. For example, as T1 is the default case mapping in LaTeX, we can set for Turkish:

```
\StartBabelCommands{turkish}{}[ot1enc, fontenc=OT1]
\SetCase
  {\uccode"10=`I\relax}
  {\lccode`I="10\relax}

\StartBabelCommands{turkish}{}[unicode, fontenc=TU EU1 EU2, charset=utf8]
\SetCase
  {\uccode`i=`İ\relax
   \uccode`ı=`I\relax}
  {\lccode`İ=`i\relax
   \lccode`I=`ı\relax}

\StartBabelCommands{turkish}{}
\SetCase
  {\uccode`i="9D\relax
   \uccode"19=`I\relax}
  {\lccode"9D=`i\relax
   \lccode`I="19\relax}

\EndBabelCommands
```

(Note the mapping for OT1 is not complete.)

\SetHyphenMap {⟨*to-lower-macros*⟩}

New 3.9g Case mapping serves in TeX for two unrelated purposes: case transforms (upper/lower) and hyphenation. \SetCase handles the former, while hyphenation is handled by \SetHyphenMap and controlled with the package option hyphenmap. So, even if internally they are based on the same TeX primitive (\lccode), babel sets them separately. There are three helper macros to be used inside \SetHyphenMap:

- \BabelLower{⟨*uccode*⟩}{⟨*lccode*⟩} is similar to \lccode but it's ignored if the char has been set and saves the original lccode to restore it when switching the language (except with hyphenmap=first).

- \BabelLowerMM{⟨*uccode-from*⟩}{⟨*uccode-to*⟩}{⟨*step*⟩}{⟨*lccode-from*⟩} loops though the given uppercase codes, using the step, and assigns them the lccode, which is also increased (MM stands for *many-to-many*).

- \BabelLowerMO{⟨*uccode-from*⟩}{⟨*uccode-to*⟩}{⟨*step*⟩}{⟨*lccode*⟩} loops though the given uppercase codes, using the step, and assigns them the lccode, which is fixed (MO stands for *many-to-one*).

An example is (which is redundant, because these assignments are done by both luatex and xetex):

```
\SetHyphenMap{\BabelLowerMM{"100}{"11F}{2}{"101}}
```

This macro is not intended to fix wrong mappings done by Unicode (which are the default in both xetex and luatex) – if an assignment is wrong, fix it directly.

4 Changes

4.1 Changes in babel version 3.9

Most of the changes in version 3.9 were related to bugs, either to fix them (there were lots), or to provide some alternatives. Even new features like `\babelhyphen` are intended to solve a certain problem (in this case, the lacking of a uniform syntax and behavior for shorthands across languages). These changes are described in this manual in the corresponding place. A selective list follows:

- `\select@language` did not set `\languagename`. This meant the language in force when auxiliary files were loaded was the one used in, for example, shorthands – if the language was german, a `\select@language{spanish}` had no effect.

- `\foreignlanguage` and `otherlanguage*` messed up `\extras<language>`. Scripts, encodings and many other things were not switched correctly.

- The `:ENC` mechanism for hyphenation patterns used the encoding of the *previous* language, not that of the language being selected.

- `'` (with `activeacute`) had the original value when writing to an auxiliary file, and things like an infinite loop can happen. It worked incorrectly with `^` (if activated) and also if deactivated.

- Active chars where not reset at the end of language options, and that lead to incompatibilities between languages.

- `\textormath` raised an error with a conditional.

- `\aliasshorthand` didn't work (or only in a few and very specific cases).

- `\l@english` was defined incorrectly (using `\let` instead of `\chardef`).

- `ldf` files not bundled with babel were not recognized when called as global options.

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).

5 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.

6 `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.

7 **Tools**

1 ⟨⟨version=3.65.2543⟩⟩
2 ⟨⟨date=2021/11/01⟩⟩

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[30]. 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 and \<..> for \noexpand applied to a built macro name (the
           latter does not define the macro if undefined to \relax, because it is created locally). The result may
           be followed by extra arguments, if necessary.

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

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

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

---

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

\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@samestring 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⟩⟩
```

7.1   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.

7.2   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%
```

7.3 `base`

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

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

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

7.4 `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*{%
```

71

```
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 %
```

7.5   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
```

Make sure the language set with 'main' is the last one.

```
415 \ifx\bbl@opt@main\@nnil\else
416   \edef\bbl@language@opts{%
417     \ifx\bbl@language@opts\@empty\else\bbl@language@opts,\fi
418       \bbl@opt@main}
419 \fi
```

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

```
420 \bbl@trace{Defining IfBabelLayout}
421 \ifx\bbl@opt@layout\@nnil
422   \newcommand\IfBabelLayout[3]{#3}%
423 \else
424   \newcommand\IfBabelLayout[1]{%
425     \@expandtwoargs\in@{.#1.}{.\bbl@opt@layout.}%
426     \ifin@
427       \expandafter\@firstoftwo
428     \else
429       \expandafter\@secondoftwo
430     \fi}
431 \fi
432 ⟨/package⟩
433 ⟨∗core⟩
```

7.6   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*.

```
434 \ifx\ldf@quit\@undefined\else
435 \endinput\fi % Same line!
436 ⟨⟨Make sure ProvidesFile is defined⟩⟩
437 \ProvidesFile{babel.def}[⟨⟨date⟩⟩ ⟨⟨version⟩⟩ Babel common definitions]
438 \ifx\AtBeginDocument\@undefined  % TODO. change test.
439   ⟨⟨Emulate LaTeX⟩⟩
440 \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.

```
441 ⟨/core⟩
442 ⟨∗package | core⟩
```

8 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.

```
443 \def\bbl@version{⟨⟨version⟩⟩}
444 \def\bbl@date{⟨⟨date⟩⟩}
445 ⟨⟨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.

```
446 \def\adddialect#1#2{%
447   \global\chardef#1#2\relax
448   \bbl@usehooks{adddialect}{{#1}{#2}}%
449   \begingroup
450     \count@#1\relax
451     \def\bbl@elt##1##2##3##4{%
452       \ifnum\count@=##2\relax
453         \edef\bbl@tempa{\expandafter\@gobbletwo\string#1}%
454         \bbl@info{Hyphen rules for '\expandafter\@gobble\bbl@tempa'
455                   set to \expandafter\string\csname l@##1\endcsname\\%
456                   (\string\language\the\count@). Reported}%
457         \def\bbl@elt####1####2####3####4{}%
458       \fi}%
459     \bbl@cs{languages}%
460   \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.

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

74

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.

```
479 \def\bbl@bcpcase#1#2#3#4\@@#5{%
480   \ifx\@empty#3%
481     \uppercase{\def#5{#1#2}}%
482   \else
483     \uppercase{\def#5{#1}}%
484     \lowercase{\edef#5{#5#2#3#4}}%
485   \fi}
486 \def\bbl@bcplookup#1-#2-#3-#4\@@{%
487   \let\bbl@bcp\relax
488   \lowercase{\def\bbl@tempa{#1}}%
489   \ifx\@empty#2%
490     \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
491   \else\ifx\@empty#3%
492     \bbl@bcpcase#2\@empty\@empty\@@\bbl@tempb
493     \IfFileExists{babel-\bbl@tempa-\bbl@tempb.ini}%
494       {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb}}%
495       {}%
496     \ifx\bbl@bcp\relax
497       \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
498     \fi
499   \else
500     \bbl@bcpcase#2\@empty\@empty\@@\bbl@tempb
501     \bbl@bcpcase#3\@empty\@empty\@@\bbl@tempc
502     \IfFileExists{babel-\bbl@tempa-\bbl@tempb-\bbl@tempc.ini}%
503       {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb-\bbl@tempc}}%
504       {}%
505     \ifx\bbl@bcp\relax
506       \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
507         {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
508         {}%
509     \fi
510     \ifx\bbl@bcp\relax
511       \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
512         {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
513         {}%
514     \fi
515     \ifx\bbl@bcp\relax
516       \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
517     \fi
518   \fi\fi}
519 \let\bbl@initoload\relax
520 \def\bbl@provide@locale{%
521   \ifx\babelprovide\@undefined
522     \bbl@error{For a language to be defined on the fly 'base'\\%
523                is not enough, and the whole package must be\\%
524                loaded. Either delete the 'base' option or\\%
525                request the languages explicitly}%
526               {See the manual for further details.}%
527   \fi
528 % TODO. Option to search if loaded, with \LocaleForEach
529   \let\bbl@auxname\languagename % Still necessary. TODO
530   \bbl@ifunset{bbl@bcp@map@\languagename}{}% Move uplevel??
531     {\edef\languagename{\@nameuse{bbl@bcp@map@\languagename}}}%
```

```
532    \ifbbl@bcpallowed
533      \expandafter\ifx\csname date\languagename\endcsname\relax
534        \expandafter
535        \bbl@bcplookup\languagename-\@empty-\@empty-\@empty\@@
536        \ifx\bbl@bcp\relax\else  % Returned by \bbl@bcplookup
537          \edef\languagename{\bbl@bcp@prefix\bbl@bcp}%
538          \edef\localename{\bbl@bcp@prefix\bbl@bcp}%
539          \expandafter\ifx\csname date\languagename\endcsname\relax
540            \let\bbl@initoload\bbl@bcp
541            \bbl@exp{\\\babelprovide[\bbl@autoload@bcpoptions]{\languagename}}%
542            \let\bbl@initoload\relax
543          \fi
544          \bbl@csarg\xdef{bcp@map@\bbl@bcp}{\localename}%
545        \fi
546      \fi
547    \fi
548    \expandafter\ifx\csname date\languagename\endcsname\relax
549      \IfFileExists{babel-\languagename.tex}%
550        {\bbl@exp{\\\babelprovide[\bbl@autoload@options]{\languagename}}}%
551        {}%
552    \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.

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

8.1   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.

```
560 \let\bbl@select@type\z@
561 \edef\selectlanguage{%
562   \noexpand\protect
563   \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.

```
564 \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.

```
565 \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.

76

\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.

```
566 \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
```
567 \def\bbl@push@language{%
568   \ifx\languagename\@undefined\else
569     \ifx\currentgrouplevel\@undefined
570       \xdef\bbl@language@stack{\languagename+\bbl@language@stack}%
571     \else
572       \ifnum\currentgrouplevel=\z@
573         \xdef\bbl@language@stack{\languagename+}%
574       \else
575         \xdef\bbl@language@stack{\languagename+\bbl@language@stack}%
576       \fi
577     \fi
578   \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.

```
579 \def\bbl@pop@lang#1+#2\@@{%
580   \edef\languagename{#1}%
581   \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).

```
582 \let\bbl@ifrestoring\@secondoftwo
583 \def\bbl@pop@language{%
584   \expandafter\bbl@pop@lang\bbl@language@stack\@@
585   \let\bbl@ifrestoring\@firstoftwo
586   \expandafter\bbl@set@language\expandafter{\languagename}%
587   \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).

```
588 \chardef\localeid\z@
589 \def\bbl@id@last{0}     % No real need for a new counter
590 \def\bbl@id@assign{%
591   \bbl@ifunset{bbl@id@@\languagename}%
592     {\count@\bbl@id@last\relax
593      \advance\count@\@ne
594      \bbl@csarg\chardef{id@@\languagename}\count@
595      \edef\bbl@id@last{\the\count@}%
596      \ifcase\bbl@engine\or
```

```
597        \directlua{
598          Babel = Babel or {}
599          Babel.locale_props = Babel.locale_props or {}
600          Babel.locale_props[bbl@id@last] = {}
601          Babel.locale_props[bbl@id@last].name = '\languagename'
602        }%
603      \fi}%
604    {}%
605    \chardef\localeid\bbl@cl{id@}}
```

The unprotected part of `\selectlanguage`.

```
606 \expandafter\def\csname selectlanguage \endcsname#1{%
607   \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\tw@\fi
608   \bbl@push@language
609   \aftergroup\bbl@pop@language
610   \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).

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

```
641      \ifx\babel@aux\@gobbletwo\else % Set if single in the first, redundant
642        \bbl@savelastskip
643        \protected@write\@auxout{}{\string\babel@aux{\bbl@auxname}{}}%
644        \bbl@restorelastskip
645      \fi
646      \bbl@usehooks{write}{}%
647    \fi
648  \fi}
649 %
650 \let\bbl@restorelastskip\relax
651 \let\bbl@savelastskip\relax
652 %
653 \newif\ifbbl@bcpallowed
654 \bbl@bcpallowedfalse
655 \def\select@language#1{% from set@, babel@aux
656   % set hymap
657   \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
658   % set name
659   \edef\languagename{#1}%
660   \bbl@fixname\languagename
661   % TODO. name@map must be here?
662   \bbl@provide@locale
663   \bbl@iflanguage\languagename{%
664      \expandafter\ifx\csname date\languagename\endcsname\relax
665       \bbl@error
666         {Unknown language '\languagename'. Either you have\\%
667          misspelled its name, it has not been installed,\\%
668          or you requested it in a previous run. Fix its name,\\%
669          install it or just rerun the file, respectively. In\\%
670          some cases, you may need to remove the aux file}%
671         {You may proceed, but expect wrong results}%
672      \else
673        % set type
674        \let\bbl@select@type\z@
675        \expandafter\bbl@switch\expandafter{\languagename}%
676      \fi}}
677 \def\babel@aux#1#2{%
678   \select@language{#1}%
679   \bbl@foreach\BabelContentsFiles{% \relax -> don't assume vertical mode
680     \@writefile{##1}{\babel@toc{#1}{#2}\relax}}}% TODO - plain?
681 \def\babel@toc#1#2{%
682   \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.

```
683 \newif\ifbbl@usedategroup
684 \def\bbl@switch#1{%  from select@, foreign@
685   % make sure there is info for the language if so requested
686   \bbl@ensureinfo{#1}%
```

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

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

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.

```
767 \long\def\otherlanguage#1{%
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    \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
777    \def\bbl@select@opts{#1}%
778    \foreign@language{#2}}
```

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

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

\foreignlanguage   The \foreignlanguage command is another substitute for the \selectlanguage command. This command takes two arguments, the first argument is the name of the language to use for typesetting the text specified in the second argument.
Unlike \selectlanguage this command doesn't switch *everything*, it only switches the hyphenation rules and the extra definitions for the language specified. It does this within a group and assumes the \extras⟨*lang*⟩ command doesn't make any \global changes. The coding is very similar to part of \selectlanguage.

\bbl@beforeforeign is a trick to fix a bug in bidi texts. \foreignlanguage is supposed to be a 'text' command, and therefore it must emit a \leavevmode, but it does not, and therefore the indent is placed on the opposite margin. For backward compatibility, however, it is done only if a right-to-left script is requested; otherwise, it is no-op.

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

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

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

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

```
806 \def\foreign@language#1{%
807   % set name
808   \edef\languagename{#1}%
809   \ifbbl@usedategroup
810     \bbl@add\bbl@select@opts{,date,}%
811     \bbl@usedategroupfalse
812   \fi
813   \bbl@fixname\languagename
814   % TODO. name@map here?
815   \bbl@provide@locale
816   \bbl@iflanguage\languagename{%
817     \expandafter\ifx\csname date\languagename\endcsname\relax
818       \bbl@warning   % TODO - why a warning, not an error?
819         {Unknown language '#1'. Either you have\\%
```

```
820          misspelled its name, it has not been installed,\\%
821          or you requested it in a previous run. Fix its name,\\%
822          install it or just rerun the file, respectively. In\\%
823          some cases, you may need to remove the aux file.\\%
824          I'll proceed, but expect wrong results.\\%
825          Reported}%
826      \fi
827      % set type
828      \let\bbl@select@type\@ne
829      \expandafter\bbl@switch\expandafter{\languagename}}}
```

\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.

```
830 \let\bbl@hyphlist\@empty
831 \let\bbl@hyphenation@\relax
832 \let\bbl@pttnlist\@empty
833 \let\bbl@patterns@\relax
834 \let\bbl@hymapsel=\@cclv
835 \def\bbl@patterns#1{%
836   \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
837       \csname l@#1\endcsname
838       \edef\bbl@tempa{#1}%
839     \else
840       \csname l@#1:\f@encoding\endcsname
841       \edef\bbl@tempa{#1:\f@encoding}%
842     \fi
843   \@expandtwoargs\bbl@usehooks{patterns}{{#1}{\bbl@tempa}}%
844   % > luatex
845   \@ifundefined{bbl@hyphenation@}{}{% Can be \relax!
846     \begingroup
847       \bbl@xin@{,\number\language,}{,\bbl@hyphlist}%
848       \ifin@\else
849         \@expandtwoargs\bbl@usehooks{hyphenation}{{#1}{\bbl@tempa}}%
850         \hyphenation{%
851           \bbl@hyphenation@
852           \@ifundefined{bbl@hyphenation@#1}%
853             \@empty
854             {\space\csname bbl@hyphenation@#1\endcsname}}%
855         \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
856       \fi
857     \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*.

```
858 \def\hyphenrules#1{%
859   \edef\bbl@tempf{#1}%
860   \bbl@fixname\bbl@tempf
861   \bbl@iflanguage\bbl@tempf{%
862     \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
863     \ifx\languageshorthands\@undefined\else
```

```
864         \languageshorthands{none}%
865     \fi
866     \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax
867       \set@hyphenmins\tw@\thr@@\relax
868     \else
869       \expandafter\expandafter\expandafter\set@hyphenmins
870       \csname\bbl@tempf hyphenmins\endcsname\relax
871     \fi}}
872 \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.

```
873 \def\providehyphenmins#1#2{%
874   \expandafter\ifx\csname #1hyphenmins\endcsname\relax
875     \@namedef{#1hyphenmins}{#2}%
876   \fi}
```

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

```
877 \def\set@hyphenmins#1#2{%
878   \lefthyphenmin#1\relax
879   \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.

```
880 \ifx\ProvidesFile\@undefined
881   \def\ProvidesLanguage#1[#2 #3 #4]{%
882     \wlog{Language: #1 #4 #3 <#2>}%
883     }
884 \else
885   \def\ProvidesLanguage#1{%
886     \begingroup
887       \catcode`\ 10 %
888       \@makeother\/%
889       \@ifnextchar[%
890         {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}}
891   \def\@provideslanguage#1[#2]{%
892     \wlog{Language: #1 #2}%
893     \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
894     \endgroup}
895 \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.

```
896 \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.

```
897 \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':

```
898 \providecommand\setlocale{%
899   \bbl@error
900     {Not yet available}%
901     {Find an armchair, sit down and wait}}
```

```
902 \let\uselocale\setlocale
903 \let\locale\setlocale
904 \let\selectlocale\setlocale
905 \let\localename\setlocale
906 \let\textlocale\setlocale
907 \let\textlanguage\setlocale
908 \let\languagetext\setlocale
```

8.2 Errors

\@nolanerr
\@nopatterns

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.

\@noopterr

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

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

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

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

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

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

```
973 \def\addto#1#2{%
974   \ifx#1\@undefined
975     \def#1{#2}%
976   \else
977     \ifx#1\relax
978       \def#1{#2}%
979     \else
980       {\toks@\expandafter{#1#2}%
981         \xdef#1{\the\toks@}}%
982     \fi
983   \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?

```
984 \def\bbl@withactive#1#2{%
985   \begingroup
986     \lccode`\~=`#2\relax
987     \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.

```
988 \def\bbl@redefine#1{%
```

86

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

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

```
993 \def\bbl@redefine@long#1{%
994  \edef\bbl@tempa{\bbl@stripslash#1}%
995  \expandafter\let\csname org@\bbl@tempa\endcsname#1%
996  \expandafter\long\expandafter\def\csname\bbl@tempa\endcsname}
997 \@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␣.

```
998 \def\bbl@redefinerobust#1{%
999   \edef\bbl@tempa{\bbl@stripslash#1}%
1000  \bbl@ifunset{\bbl@tempa\space}%
1001    {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
1002     \bbl@exp{\def\\#1{\\\protect\<\bbl@tempa\space>}}}%
1003    {\bbl@exp{\let\<org@\bbl@tempa>\<\bbl@tempa\space>}}%
1004    \@namedef{\bbl@tempa\space}}
1005 \@onlypreamble\bbl@redefinerobust
```

8.3 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.

```
1006 \bbl@trace{Hooks}
1007 \newcommand\AddBabelHook[3][]{%
1008  \bbl@ifunset{bbl@hk@#2}{\EnableBabelHook{#2}}{}%
1009  \def\bbl@tempa##1,#3=##2,##3\@empty{\def\bbl@tempb{##2}}%
1010  \expandafter\bbl@tempa\bbl@evargs,#3=,\@empty
1011  \bbl@ifunset{bbl@ev@#2@#3@#1}%
1012    {\bbl@csarg\bbl@add{ev@#3@#1}{\bbl@elth{#2}}}%
1013    {\bbl@csarg\let{ev@#2@#3@#1}\relax}%
1014  \bbl@csarg\newcommand{ev@#2@#3@#1}[\bbl@tempb]}
1015 \newcommand\EnableBabelHook[1]{\bbl@csarg\let{hk@#1}\@firstofone}
1016 \newcommand\DisableBabelHook[1]{\bbl@csarg\let{hk@#1}\@gobble}
1017 \def\bbl@usehooks#1#2{%
1018  \ifx\UseHook\@undefined\else\UseHook{babel/*/#1}\fi
1019  \def\bbl@elth##1{%
1020    \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#1@}#2}}%
1021  \bbl@cs{ev@#1@}%
1022  \ifx\languagename\@undefined\else % Test required for Plain (?)
1023    \ifx\UseHook\@undefined\else\UseHook{babel/\languagename/#1}\fi
1024    \def\bbl@elth##1{%
1025      \bbl@cs{hk@##1}{\bbl@cl{ev@##1@#1}#2}}%
1026    \bbl@cl{ev@#1}%
1027  \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).

```
1028 \def\bbl@evargs{,% <- don't delete this comma
1029   everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
```

```
1030    adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
1031    beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
1032    hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%
1033    beforestart=0,languagename=2}
1034 \ifx\NewHook\@undefined\else
1035    \def\bbl@tempa#1=#2\@@{\NewHook{babel/#1}}
1036    \bbl@foreach\bbl@evargs{\bbl@tempa#1\@@}
1037 \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 \fontcoding 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.

```
1038 \bbl@trace{Defining babelensure}
1039 \newcommand\babelensure[2][]{%  TODO - revise test files
1040    \AddBabelHook{babel-ensure}{afterextras}{%
1041       \ifcase\bbl@select@type
1042          \bbl@cl{e}%
1043       \fi}%
1044    \begingroup
1045       \let\bbl@ens@include\@empty
1046       \let\bbl@ens@exclude\@empty
1047       \def\bbl@ens@fontenc{\relax}%
1048       \def\bbl@tempb##1{%
1049          \ifx\@empty##1\else\noexpand##1\expandafter\bbl@tempb\fi}%
1050       \edef\bbl@tempa{\bbl@tempb#1\@empty}%
1051       \def\bbl@tempb##1=##2\@@{\@namedef{bbl@ens@##1}{##2}}%
1052       \bbl@foreach\bbl@tempa{\bbl@tempb##1\@@}%
1053       \def\bbl@tempc{\bbl@ensure}%
1054       \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1055          \expandafter{\bbl@ens@include}}%
1056       \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1057          \expandafter{\bbl@ens@exclude}}%
1058       \toks@\expandafter{\bbl@tempc}%
1059       \bbl@exp{%
1060    \endgroup
1061    \def\<bbl@e@#2>{\the\toks@{\bbl@ens@fontenc}}}}
1062 \def\bbl@ensure#1#2#3{% 1: include 2: exclude 3: fontenc
1063    \def\bbl@tempb##1{% elt for (excluding) \bbl@captionslist list
1064       \ifx##1\@undefined % 3.32 - Don't assume the macro exists
1065          \edef##1{\noexpand\bbl@nocaption
1066             {\bbl@stripslash##1}{\languagename\bbl@stripslash##1}}%
1067       \fi
1068       \ifx##1\@empty\else
1069          \in@{##1}{#2}%
1070          \ifin@\else
1071             \bbl@ifunset{bbl@ensure@\languagename}%
1072                {\bbl@exp{%
1073                   \\\DeclareRobustCommand\<bbl@ensure@\languagename>[1]{%
1074                      \\\foreignlanguage{\languagename}%
1075                      {\ifx\relax#3\else
1076                         \\\fontencoding{#3}\\\selectfont
1077                       \fi
1078                       ########1}}}}%
```

88

```
1079            {}%
1080          \toks@\expandafter{##1}%
1081          \edef##1{%
1082             \bbl@csarg\noexpand{ensure@\languagename}%
1083             {\the\toks@}}%
1084        \fi
1085        \expandafter\bbl@tempb
1086      \fi}%
1087    \expandafter\bbl@tempb\bbl@captionslist\today\@empty
1088    \def\bbl@tempa##1{% elt for include list
1089      \ifx##1\@empty\else
1090        \bbl@csarg\in@{ensure@\languagename\expandafter}\expandafter{##1}%
1091        \ifin@\else
1092          \bbl@tempb##1\@empty
1093        \fi
1094        \expandafter\bbl@tempa
1095      \fi}%
1096    \bbl@tempa#1\@empty}
1097  \def\bbl@captionslist{%
1098    \prefacename\refname\abstractname\bibname\chaptername\appendixname
1099    \contentsname\listfigurename\listtablename\indexname\figurename
1100    \tablename\partname\enclname\ccname\headtoname\pagename\seename
1101    \alsoname\proofname\glossaryname}
```

8.4   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.

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

```
1116   \chardef\eqcatcode=\catcode`\=
1117   \catcode`\==12\relax
1118   \expandafter\if\expandafter\@backslashchar
1119              \expandafter\@car\string#2\@nil
1120     \ifx#2\@undefined\else
1121       \ldf@quit{#1}%
1122     \fi
1123   \else
1124     \expandafter\ifx\csname#2\endcsname\relax\else
1125       \ldf@quit{#1}%
1126     \fi
1127   \fi
1128   \bbl@ldfinit}
```

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

```
1129 \def\ldf@quit#1{%
1130   \expandafter\main@language\expandafter{#1}%
1131   \catcode`\@=\atcatcode \let\atcatcode\relax
1132   \catcode`\==\eqcatcode \let\eqcatcode\relax
1133   \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.

```
1134 \def\bbl@afterldf#1{% TODO. Merge into the next macro? Unused elsewhere
1135   \bbl@afterlang
1136   \let\bbl@afterlang\relax
1137   \let\BabelModifiers\relax
1138   \let\bbl@screset\relax}%
1139 \def\ldf@finish#1{%
1140   \loadlocalcfg{#1}%
1141   \bbl@afterldf{#1}%
1142   \expandafter\main@language\expandafter{#1}%
1143   \catcode`\@=\atcatcode \let\atcatcode\relax
1144   \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.

```
1145 \@onlypreamble\LdfInit
1146 \@onlypreamble\ldf@quit
1147 \@onlypreamble\ldf@finish
```

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

```
1148 \def\main@language#1{%
1149   \def\bbl@main@language{#1}%
1150   \let\languagename\bbl@main@language % TODO. Set localename
1151   \bbl@id@assign
1152   \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.

```
1153 \def\bbl@beforestart{%
1154   \def\@nolanerr##1{%
1155     \bbl@warning{Undefined language '##1' in aux.\\Reported}}%
```

90

```
1156    \bbl@usehooks{beforestart}{}%
1157    \global\let\bbl@beforestart\relax}
1158 \AtBeginDocument{%
1159    {\@nameuse{bbl@beforestart}}%  Group!
1160    \if@filesw
1161      \providecommand\babel@aux[2]{}%
1162      \immediate\write\@mainaux{%
1163        \string\providecommand\string\babel@aux[2]{}}%
1164      \immediate\write\@mainaux{\string\@nameuse{bbl@beforestart}}%
1165    \fi
1166    \expandafter\selectlanguage\expandafter{\bbl@main@language}%
1167    \ifbbl@single  % must go after the line above.
1168      \renewcommand\selectlanguage[1]{}%
1169      \renewcommand\foreignlanguage[2]{#2}%
1170      \global\let\babel@aux\@gobbletwo  % Also as flag
1171    \fi
1172    \ifcase\bbl@engine\or\pagedir\bodydir\fi} % TODO - a better place
```

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

```
1173 \def\select@language@x#1{%
1174    \ifcase\bbl@select@type
1175      \bbl@ifsamestring\languagename{#1}{}{\select@language{#1}}%
1176    \else
1177      \select@language{#1}%
1178    \fi}
```

8.5   Shorthands

<code>\bbl@add@special</code>   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.

```
1179 \bbl@trace{Shorhands}
1180 \def\bbl@add@special#1{% 1:a macro like \", \?, etc.
1181    \bbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
1182    \bbl@ifunset{@sanitize}{}{\bbl@add\@sanitize{\@makeother#1}}%
1183    \ifx\nfss@catcodes\@undefined\else % TODO - same for above
1184      \begingroup
1185        \catcode`#1\active
1186        \nfss@catcodes
1187        \ifnum\catcode`#1=\active
1188          \endgroup
1189          \bbl@add\nfss@catcodes{\@makeother#1}%
1190        \else
1191          \endgroup
1192        \fi
1193    \fi}
```

<code>\bbl@remove@special</code>   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.

```
1194 \def\bbl@remove@special#1{%
1195    \begingroup
1196      \def\x##1##2{\ifnum`#1=`##2\noexpand\@empty
1197                    \else\noexpand##1\noexpand##2\fi}%
1198      \def\do{\x\do}%
1199      \def\@makeother{\x\@makeother}%
```

91

```
1200    \edef\x{\endgroup
1201      \def\noexpand\dospecials{\dospecials}%
1202      \expandafter\ifx\csname @sanitize\endcsname\relax\else
1203        \def\noexpand\@sanitize{\@sanitize}%
1204      \fi}%
1205    \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).

```
1206 \def\bbl@active@def#1#2#3#4{%
1207   \@namedef{#3#1}{%
1208     \expandafter\ifx\csname#2@sh@#1@\endcsname\relax
1209       \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
1210     \else
1211       \bbl@afterfi\csname#2@sh@#1@\endcsname
1212     \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.

```
1213   \long\@namedef{#3@arg#1}##1{%
1214     \expandafter\ifx\csname#2@sh@#1@\string##1@\endcsname\relax
1215       \bbl@afterelse\csname#4#1\endcsname##1%
1216     \else
1217       \bbl@afterfi\csname#2@sh@#1@\string##1@\endcsname
1218     \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.

```
1219 \def\initiate@active@char#1{%
1220   \bbl@ifunset{active@char\string#1}%
1221     {\bbl@withactive
1222       {\expandafter\@initiate@active@char\expandafter}#1\string#1#1}%
1223     {}}
```

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).

```
1224 \def\@initiate@active@char#1#2#3{%
1225   \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
1226   \ifx#1\@undefined
1227     \bbl@csarg\def{oridef@#2}{\def#1{\active@prefix#1\@undefined}}%
1228   \else
```

```
1229    \bbl@csarg\let{oridef@@#2}#1%
1230    \bbl@csarg\edef{oridef@#2}{%
1231      \let\noexpand#1%
1232      \expandafter\noexpand\csname bbl@oridef@@#2\endcsname}%
1233  \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*).

```
1234    \ifx#1#3\relax
1235      \expandafter\let\csname normal@char#2\endcsname#3%
1236    \else
1237      \bbl@info{Making #2 an active character}%
1238      \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
1239        \@namedef{normal@char#2}{%
1240          \textormath{#3}{\csname bbl@oridef@@#2\endcsname}}%
1241      \else
1242        \@namedef{normal@char#2}{#3}%
1243      \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).

```
1244      \bbl@restoreactive{#2}%
1245      \AtBeginDocument{%
1246        \catcode`#2\active
1247        \if@filesw
1248          \immediate\write\@mainaux{\catcode`\string#2\active}%
1249        \fi}%
1250      \expandafter\bbl@add@special\csname#2\endcsname
1251      \catcode`#2\active
1252    \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⟩).

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

```
1269        \noexpand\fi}%
1270      {\expandafter\noexpand\csname normal@char#2\endcsname}}%
1271    \bbl@csarg\edef{doactive#2}{%
1272      \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

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

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

```
1273    \bbl@csarg\edef{active@#2}{%
1274      \noexpand\active@prefix\noexpand#1%
1275      \expandafter\noexpand\csname active@char#2\endcsname}%
1276    \bbl@csarg\edef{normal@#2}{%
1277      \noexpand\active@prefix\noexpand#1%
1278      \expandafter\noexpand\csname normal@char#2\endcsname}%
1279    \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.

```
1280    \bbl@active@def#2\user@group{user@active}{language@active}%
1281    \bbl@active@def#2\language@group{language@active}{system@active}%
1282    \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.

```
1283    \expandafter\edef\csname\user@group @sh@#2@@\endcsname
1284      {\expandafter\noexpand\csname normal@char#2\endcsname}%
1285    \expandafter\edef\csname\user@group @sh@#2@\string\protect@\endcsname
1286      {\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.

```
1287    \if\string'#2%
1288      \let\prim@s\bbl@prim@s
1289      \let\active@math@prime#1%
1290    \fi
1291    \bbl@usehooks{initiateactive}{{#1}{#2}{#3}}}
```

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

```
1292 ⟨⟨∗More package options⟩⟩ ≡
1293 \DeclareOption{math=active}{}
1294 \DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
1295 ⟨⟨/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.

```
1296 \@ifpackagewith{babel}{KeepShorthandsActive}%
1297    {\let\bbl@restoreactive\@gobble}%
1298    {\def\bbl@restoreactive#1{%
1299      \bbl@exp{%
```

94

```
1300        \\\AfterBabelLanguage\\\CurrentOption
1301          {\catcode`#1=\the\catcode`#1\relax}%
1302        \\\\AtEndOfPackage
1303          {\catcode`#1=\the\catcode`#1\relax}}}%
1304      \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.

```
1305 \def\bbl@sh@select#1#2{%
1306   \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
1307     \bbl@afterelse\bbl@scndcs
1308   \else
1309     \bbl@afterfi\csname#1@sh@#2@sel\endcsname
1310   \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.

```
1311 \begingroup
1312 \bbl@ifunset{ifincsname}% TODO. Ugly. Correct? Only Plain?
1313   {\gdef\active@prefix#1{%
1314       \ifx\protect\@typeset@protect
1315       \else
1316         \ifx\protect\@unexpandable@protect
1317           \noexpand#1%
1318         \else
1319           \protect#1%
1320         \fi
1321         \expandafter\@gobble
1322       \fi}}
1323   {\gdef\active@prefix#1{%
1324       \ifincsname
1325         \string#1%
1326         \expandafter\@gobble
1327       \else
1328         \ifx\protect\@typeset@protect
1329         \else
1330           \ifx\protect\@unexpandable@protect
1331             \noexpand#1%
1332           \else
1333             \protect#1%
1334           \fi
1335           \expandafter\expandafter\expandafter\@gobble
1336         \fi
1337     \fi}}
1338 \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⟩.

```
1339 \newif\if@safe@actives
1340 \@safe@activesfalse
```

95

\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.

```
1341 \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.

```
1342 \chardef\bbl@activated\z@
1343 \def\bbl@activate#1{%
1344   \chardef\bbl@activated\@ne
1345   \bbl@withactive{\expandafter\let\expandafter}#1%
1346     \csname bbl@active@\string#1\endcsname}
1347 \def\bbl@deactivate#1{%
1348   \chardef\bbl@activated\tw@
1349   \bbl@withactive{\expandafter\let\expandafter}#1%
1350     \csname bbl@normal@\string#1\endcsname}
```

\bbl@firstcs
\bbl@scndcs   These macros are used only as a trick when declaring shorthands.

```
1351 \def\bbl@firstcs#1#2{\csname#1\endcsname}
1352 \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.

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

```
1377       \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%
1378         {\def\bbl@tempa{#4}%
1379          \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
1380          \else
1381            \bbl@info
1382              {Redefining #1 shorthand \string#2\string#3\\%
1383               in language \CurrentOption}%
1384          \fi}%
1385       \@namedef{#1@sh@\string#2@\string#3@}{#4}%
1386    \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.

```
1387 \def\textormath{%
1388    \ifmmode
1389      \expandafter\@secondoftwo
1390    \else
1391      \expandafter\@firstoftwo
1392    \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'.

```
1393 \def\user@group{user}
1394 \def\language@group{english} % TODO. I don't like defaults
1395 \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.

```
1396 \def\useshorthands{%
1397    \@ifstar\bbl@usesh@s{\bbl@usesh@x{}}}
1398 \def\bbl@usesh@s#1{%
1399    \bbl@usesh@x
1400      {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bbl@activate{#1}}}%
1401      {#1}}
1402 \def\bbl@usesh@x#1#2{%
1403    \bbl@ifshorthand{#2}%
1404      {\def\user@group{user}%
1405       \initiate@active@char{#2}%
1406       #1%
1407       \bbl@activate{#2}}%
1408      {\bbl@error
1409         {I can't declare a shorthand turned off (\string#2)}
1410         {Sorry, but you can't use shorthands which have been\\%
1411          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.

```
1412 \def\user@language@group{user@\language@group}
1413 \def\bbl@set@user@generic#1#2{%
1414    \bbl@ifunset{user@generic@active#1}%
1415      {\bbl@active@def#1\user@language@group{user@active}{user@generic@active}%
1416       \bbl@active@def#1\user@group{user@generic@active}{language@active}%
1417       \expandafter\edef\csname#2@sh@#1@@\endcsname{%
```

97

```
1418        \expandafter\noexpand\csname normal@char#1\endcsname}%
1419      \expandafter\edef\csname#2@sh@#1@\string\protect@\endcsname{%
1420        \expandafter\noexpand\csname user@active#1\endcsname}}%
1421    \@empty}
1422 \newcommand\defineshorthand[3][user]{%
1423    \edef\bbl@tempa{\zap@space#1 \@empty}%
1424    \bbl@for\bbl@tempb\bbl@tempa{%
1425      \if*\expandafter\@car\bbl@tempb\@nil
1426        \edef\bbl@tempb{user@\expandafter\@gobble\bbl@tempb}%
1427        \@expandtwoargs
1428          \bbl@set@user@generic{\expandafter\string\@car#2\@nil}\bbl@tempb
1429      \fi
1430      \declare@shorthand{\bbl@tempb}{#2}{#3}}}
```

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].

```
1431 \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".

```
1432 \def\aliasshorthand#1#2{%
1433    \bbl@ifshorthand{#2}%
1434      {\expandafter\ifx\csname active@char\string#2\endcsname\relax
1435        \ifx\document\@notprerr
1436          \@notshorthand{#2}%
1437        \else
1438          \initiate@active@char{#2}%
1439          \expandafter\let\csname active@char\string#2\expandafter\endcsname
1440            \csname active@char\string#1\endcsname
1441          \expandafter\let\csname normal@char\string#2\expandafter\endcsname
1442            \csname normal@char\string#1\endcsname
1443          \bbl@activate{#2}%
1444        \fi
1445      \fi}%
1446      {\bbl@error
1447        {Cannot declare a shorthand turned off (\string#2)}
1448        {Sorry, but you cannot use shorthands which have been\\%
1449         turned off in the package options}}}
```

\@notshorthand

```
1450 \def\@notshorthand#1{%
1451    \bbl@error{%
1452      The character '\string #1' should be made a shorthand character;\\%
1453      add the command \string\useshorthands\string{#1\string} to
1454      the preamble.\\%
1455      I will ignore your instruction}%
1456      {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.

```
1457 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
1458 \DeclareRobustCommand*\shorthandoff{%
1459    \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
1460 \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.

```
1461 \def\bbl@switch@sh#1#2{%
1462   \ifx#2\@nnil\else
1463     \bbl@ifunset{bbl@active@\string#2}%
1464       {\bbl@error
1465         {I can't switch '\string#2' on or off--not a shorthand}%
1466         {This character is not a shorthand. Maybe you made\\%
1467          a typing mistake? I will ignore your instruction.}}%
1468       {\ifcase#1%   off, on, off*
1469         \catcode`#212\relax
1470        \or
1471         \catcode`#2\active
1472         \bbl@ifunset{bbl@shdef@\string#2}%
1473           {}%
1474           {\bbl@withactive{\expandafter\let\expandafter}#2%
1475              \csname bbl@shdef@\string#2\endcsname
1476            \bbl@csarg\let{shdef@\string#2}\relax}%
1477         \ifcase\bbl@activated\or
1478           \bbl@activate{#2}%
1479         \else
1480           \bbl@deactivate{#2}%
1481         \fi
1482        \or
1483         \bbl@ifunset{bbl@shdef@\string#2}%
1484           {\bbl@withactive{\bbl@csarg\let{shdef@\string#2}}#2}%
1485           {}%
1486         \csname bbl@oricat@\string#2\endcsname
1487         \csname bbl@oridef@\string#2\endcsname
1488       \fi}%
1489     \bbl@afterfi\bbl@switch@sh#1%
1490   \fi}
```

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

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

```
1512    \let\bbl@s@deactivate\bbl@deactivate
1513    \def\bbl@deactivate#1{%
1514      \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
1515  \fi
```

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

```
1516  \newcommand\ifbabelshorthand[3]{\bbl@ifunset{bbl@active@\string#1}{#3}{#2}}
```

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

```
1517  \def\bbl@prim@s{%
1518    \prime\futurelet\@let@token\bbl@pr@m@s}
1519  \def\bbl@if@primes#1#2{%
1520    \ifx#1\@let@token
1521      \expandafter\@firstoftwo
1522    \else\ifx#2\@let@token
1523      \bbl@afterelse\expandafter\@firstoftwo
1524    \else
1525      \bbl@afterfi\expandafter\@secondoftwo
1526    \fi\fi}
1527  \begingroup
1528    \catcode`\^=7  \catcode`\*=\active  \lccode`\*=`\^
1529    \catcode`\'=12 \catcode`\"=\active  \lccode`\"=`\'
1530    \lowercase{%
1531      \gdef\bbl@pr@m@s{%
1532        \bbl@if@primes"'%
1533          \pr@@@s
1534          {\bbl@if@primes*^\pr@@@t\egroup}}}
1535  \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).

```
1536  \initiate@active@char{~}
1537  \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
1538  \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.

```
1539  \expandafter\def\csname OT1dqpos\endcsname{127}
1540  \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

```
1541  \ifx\f@encoding\@undefined
1542    \def\f@encoding{OT1}
1543  \fi
```

8.6 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.

```
1544 \bbl@trace{Language attributes}
1545 \newcommand\languageattribute[2]{%
1546   \def\bbl@tempc{#1}%
1547   \bbl@fixname\bbl@tempc
1548   \bbl@iflanguage\bbl@tempc{%
1549     \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.

```
1550       \ifx\bbl@known@attribs\@undefined
1551         \in@false
1552       \else
1553         \bbl@xin@{,\bbl@tempc-##1,}{,\bbl@known@attribs,}%
1554       \fi
1555       \ifin@
1556         \bbl@warning{%
1557           You have more than once selected the attribute '##1'\\%
1558           for language #1. Reported}%
1559       \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.

```
1560         \bbl@exp{%
1561           \\\bbl@add@list\\\bbl@known@attribs{\bbl@tempc-##1}}%
1562         \edef\bbl@tempa{\bbl@tempc-##1}%
1563         \expandafter\bbl@ifknown@ttrib\expandafter{\bbl@tempa}\bbl@attributes%
1564         {\csname\bbl@tempc @attr@##1\endcsname}%
1565         {\@attrerr{\bbl@tempc}{##1}}%
1566       \fi}}}
1567 \@onlypreamble\languageattribute
```

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

```
1568 \newcommand*{\@attrerr}[2]{%
1569   \bbl@error
1570     {The attribute #2 is unknown for language #1.}%
1571     {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}.

```
1572 \def\bbl@declare@ttribute#1#2#3{%
1573   \bbl@xin@{,#2,}{,\BabelModifiers,}%
1574   \ifin@
1575     \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
1576   \fi
1577   \bbl@add@list\bbl@attributes{#1-#2}%
1578   \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.

```
1579 \def\bbl@ifattributeset#1#2#3#4{%
1580   \ifx\bbl@known@attribs\@undefined
1581     \in@false
1582   \else
1583     \bbl@xin@{,#1-#2,}{,\bbl@known@attribs,}%
1584   \fi
1585   \ifin@
1586     \bbl@afterelse#3%
1587   \else
1588     \bbl@afterfi#4%
1589   \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.

```
1590 \def\bbl@ifknown@ttrib#1#2{%
1591   \let\bbl@tempa\@secondoftwo
1592   \bbl@loopx\bbl@tempb{#2}{%
1593     \expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{,#1,}%
1594     \ifin@
1595       \let\bbl@tempa\@firstoftwo
1596     \else
1597     \fi}%
1598   \bbl@tempa}
```

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

```
1599 \def\bbl@clear@ttribs{%
1600   \ifx\bbl@attributes\@undefined\else
1601     \bbl@loopx\bbl@tempa{\bbl@attributes}{%
1602       \expandafter\bbl@clear@ttrib\bbl@tempa.
1603       }%
1604     \let\bbl@attributes\@undefined
1605   \fi}
1606 \def\bbl@clear@ttrib#1-#2.{%
1607   \expandafter\let\csname#1@attr@#2\endcsname\@undefined}
1608 \AtBeginDocument{\bbl@clear@ttribs}
```

8.7 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

```
1609 \bbl@trace{Macros for saving definitions}
1610 \def\babel@beginsave{\babel@savecnt\z@}
```

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

```
1611 \newcount\babel@savecnt
1612 \babel@beginsave
```

\babel@save   The macro `\babel@save`⟨*csname*⟩ saves the current meaning of the control sequence ⟨*csname*⟩ to
\babel@savevariable   `\originalTeX`[31]. To do this, we let the current meaning to a temporary control sequence, the restore
commands are appended to `\originalTeX` and the counter is incremented. The macro
`\babel@savevariable`⟨*variable*⟩ saves the value of the variable. ⟨*variable*⟩ can be anything allowed
after the `\the` primitive.

```
1613 \def\babel@save#1{%
1614   \expandafter\let\csname babel@\number\babel@savecnt\endcsname#1\relax
1615   \toks@\expandafter{\originalTeX\let#1=}%
1616   \bbl@exp{%
1617     \def\\\originalTeX{\the\toks@\<babel@\number\babel@savecnt>\relax}}%
1618   \advance\babel@savecnt\@ne}
1619 \def\babel@savevariable#1{%
1620   \toks@\expandafter{\originalTeX #1=}%
1621   \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.

```
1622 \def\bbl@frenchspacing{%
1623   \ifnum\the\sfcode`\.=\@m
1624     \let\bbl@nonfrenchspacing\relax
1625   \else
1626     \frenchspacing
1627     \let\bbl@nonfrenchspacing\nonfrenchspacing
1628   \fi}
1629 \let\bbl@nonfrenchspacing\nonfrenchspacing
1630 \let\bbl@elt\relax
1631 \edef\bbl@fs@chars{%
1632   \bbl@elt{\string.}\@m{3000}\bbl@elt{\string?}\@m{3000}%
1633   \bbl@elt{\string!}\@m{3000}\bbl@elt{\string:}\@m{2000}%
1634   \bbl@elt{\string;}\@m{1500}\bbl@elt{\string,}\@m{1250}}
1635 \def\bbl@pre@fs{%
1636   \def\bbl@elt##1##2##3{\sfcode`##1=\the\sfcode`##1\relax}%
1637   \edef\bbl@save@sfcodes{\bbl@fs@chars}}%
1638 \def\bbl@post@fs{%
1639   \bbl@save@sfcodes
1640   \edef\bbl@tempa{\bbl@cl{frspc}}%
1641   \edef\bbl@tempa{\expandafter\@car\bbl@tempa\@nil}%
1642   \if u\bbl@tempa           % do nothing
1643   \else\if n\bbl@tempa      % non french
1644     \def\bbl@elt##1##2##3{%
1645       \ifnum\sfcode`##1=##2\relax
1646         \babel@savevariable{\sfcode`##1}%
1647         \sfcode`##1=##3\relax
1648       \fi}%
1649     \bbl@fs@chars
1650   \else\if y\bbl@tempa      % french
1651     \def\bbl@elt##1##2##3{%
1652       \ifnum\sfcode`##1=##3\relax
1653         \babel@savevariable{\sfcode`##1}%
1654         \sfcode`##1=##2\relax
1655       \fi}%
1656     \bbl@fs@chars
1657   \fi\fi\fi}
```

---

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

8.8 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.

```
1658 \bbl@trace{Short tags}
1659 \def\babeltags#1{%
1660   \edef\bbl@tempa{\zap@space#1 \@empty}%
1661   \def\bbl@tempb##1=##2\@@{%
1662     \edef\bbl@tempc{%
1663       \noexpand\newcommand
1664       \expandafter\noexpand\csname ##1\endcsname{%
1665         \noexpand\protect
1666         \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}%
1667       \noexpand\newcommand
1668       \expandafter\noexpand\csname text##1\endcsname{%
1669         \noexpand\foreignlanguage{##2}}}%
1670     \bbl@tempc}%
1671   \bbl@for\bbl@tempa\bbl@tempa{%
1672     \expandafter\bbl@tempb\bbl@tempa\@@}}
```

8.9 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.

```
1673 \bbl@trace{Hyphens}
1674 \@onlypreamble\babelhyphenation
1675 \AtEndOfPackage{%
1676   \newcommand\babelhyphenation[2][\@empty]{%
1677     \ifx\bbl@hyphenation@\relax
1678       \let\bbl@hyphenation@\@empty
1679     \fi
1680     \ifx\bbl@hyphlist\@empty\else
1681       \bbl@warning{%
1682         You must not intermingle \string\selectlanguage\space and\\%
1683         \string\babelhyphenation\space or some exceptions will not\\%
1684         be taken into account. Reported}%
1685     \fi
1686     \ifx\@empty#1%
1687       \protected@edef\bbl@hyphenation@{\bbl@hyphenation@\space#2}%
1688     \else
1689       \bbl@vforeach{#1}{%
1690         \def\bbl@tempa{##1}%
1691         \bbl@fixname\bbl@tempa
1692         \bbl@iflanguage\bbl@tempa{%
1693           \bbl@csarg\protected@edef{hyphenation@\bbl@tempa}{%
1694             \bbl@ifunset{bbl@hyphenation@\bbl@tempa}%
1695               {}%
1696               {\csname bbl@hyphenation@\bbl@tempa\endcsname\space}%
1697             #2}}}%
1698     \fi}}
```

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

```
1699 \def\bbl@allowhyphens{\ifvmode\else\nobreak\hskip\z@skip\fi}
```

---

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

```
1700 \def\bbl@t@one{T1}
1701 \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.

```
1702 \newcommand\babelnullhyphen{\char\hyphenchar\font}
1703 \def\babelhyphen{\active@prefix\babelhyphen\bbl@hyphen}
1704 \def\bbl@hyphen{%
1705   \@ifstar{\bbl@hyphen@i @}{\bbl@hyphen@i\@empty}}
1706 \def\bbl@hyphen@i#1#2{%
1707   \bbl@ifunset{bbl@hy@#1#2\@empty}%
1708     {\csname bbl@#1usehyphen\endcsname{\discretionary{#2}{}{#2}}}%
1709     {\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.

```
1710 \def\bbl@usehyphen#1{%
1711   \leavevmode
1712   \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
1713   \nobreak\hskip\z@skip}
1714 \def\bbl@@usehyphen#1{%
1715   \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}
```

The following macro inserts the hyphen char.

```
1716 \def\bbl@hyphenchar{%
1717   \ifnum\hyphenchar\font=\m@ne
1718     \babelnullhyphen
1719   \else
1720     \char\hyphenchar\font
1721   \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.

```
1722 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}{}}}
1723 \def\bbl@hy@@soft{\bbl@@usehyphen{\discretionary{\bbl@hyphenchar}{}{}}}
1724 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
1725 \def\bbl@hy@@hard{\bbl@@usehyphen\bbl@hyphenchar}
1726 \def\bbl@hy@nobreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}}
1727 \def\bbl@hy@@nobreak{\mbox{\bbl@hyphenchar}}
1728 \def\bbl@hy@repeat{%
1729   \bbl@usehyphen{%
1730     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
1731 \def\bbl@hy@@repeat{%
1732   \bbl@@usehyphen{%
1733     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
1734 \def\bbl@hy@empty{\hskip\z@skip}
1735 \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.

```
1736 \def\bbl@disc#1#2{\nobreak\discretionary{#2-}{}{#1}\bbl@allowhyphens}
```

8.10 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.

```
1737 \bbl@trace{Multiencoding strings}
1738 \def\bbl@toglobal#1{\global\let#1#1}
1739 \def\bbl@recatcode#1{% TODO. Used only once?
1740   \@tempcnta="7F
1741   \def\bbl@tempa{%
1742     \ifnum\@tempcnta>"FF\else
1743       \catcode\@tempcnta=#1\relax
1744       \advance\@tempcnta\@ne
1745       \expandafter\bbl@tempa
1746     \fi}%
1747   \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).

```
1748 \@ifpackagewith{babel}{nocase}%
1749   {\let\bbl@patchuclc\relax}%
1750   {\def\bbl@patchuclc{%
1751     \global\let\bbl@patchuclc\relax
1752     \g@addto@macro\@uclclist{\reserved@b{\reserved@b\bbl@uclc}}%
1753     \gdef\bbl@uclc##1{%
1754       \let\bbl@encoded\bbl@encoded@uclc
1755       \bbl@ifunset{\languagename @bbl@uclc}% and resumes it
1756         {##1}%
1757         {\let\bbl@tempa##1\relax % Used by LANG@bbl@uclc
1758          \csname\languagename @bbl@uclc\endcsname}%
1759       {\bbl@tolower\@empty}{\bbl@toupper\@empty}}%
1760     \gdef\bbl@tolower{\csname\languagename @bbl@lc\endcsname}%
1761     \gdef\bbl@toupper{\csname\languagename @bbl@uc\endcsname}}}
```

```
1762 ⟨⟨∗More package options⟩⟩ ≡
1763 \DeclareOption{nocase}{}
1764 ⟨⟨/More package options⟩⟩
```

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

```
1765 ⟨⟨∗More package options⟩⟩ ≡
1766 \let\bbl@opt@strings\@nnil % accept strings=value
1767 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
1768 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
1769 \def\BabelStringsDefault{generic}
1770 ⟨⟨/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.

```
1771 \@onlypreamble\StartBabelCommands
1772 \def\StartBabelCommands{%
1773   \begingroup
1774   \bbl@recatcode{11}%
1775   ⟨⟨Macros local to BabelCommands⟩⟩
1776   \def\bbl@provstring##1##2{%
1777     \providecommand##1{##2}%
1778     \bbl@toglobal##1}%
1779   \global\let\bbl@scafter\@empty
1780   \let\StartBabelCommands\bbl@startcmds
1781   \ifx\BabelLanguages\relax
1782     \let\BabelLanguages\CurrentOption
1783   \fi
1784   \begingroup
1785   \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
1786   \StartBabelCommands}
1787 \def\bbl@startcmds{%
1788   \ifx\bbl@screset\@nnil\else
1789     \bbl@usehooks{stopcommands}{}%
1790   \fi
1791   \endgroup
1792   \begingroup
1793   \@ifstar
1794     {\ifx\bbl@opt@strings\@nnil
1795         \let\bbl@opt@strings\BabelStringsDefault
1796      \fi
1797      \bbl@startcmds@i}%
1798     \bbl@startcmds@i}
1799 \def\bbl@startcmds@i#1#2{%
1800   \edef\bbl@L{\zap@space#1 \@empty}%
1801   \edef\bbl@G{\zap@space#2 \@empty}%
1802   \bbl@startcmds@ii}
1803 \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.

```
1804 \newcommand\bbl@startcmds@ii[1][\@empty]{%
1805   \let\SetString\@gobbletwo
1806   \let\bbl@stringdef\@gobbletwo
1807   \let\AfterBabelCommands\@gobble
1808   \ifx\@empty#1%
1809     \def\bbl@sc@label{generic}%
1810     \def\bbl@encstring##1##2{%
1811       \ProvideTextCommandDefault##1{##2}%
1812       \bbl@toglobal##1%
1813       \expandafter\bbl@toglobal\csname\string?\string##1\endcsname}%
1814     \let\bbl@sctest\in@true
1815   \else
1816     \let\bbl@sc@charset\space % <- zapped below
```

```
1817     \let\bbl@sc@fontenc\space % <-    "        "
1818     \def\bbl@tempa##1=##2\@nil{%
1819       \bbl@csarg\edef{sc@\zap@space##1 \@empty}{##2 }}%
1820     \bbl@vforeach{label=#1}{\bbl@tempa##1\@nil}%
1821     \def\bbl@tempa##1 ##2{% space -> comma
1822       ##1%
1823       \ifx\@empty##2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa##2\fi}%
1824     \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%
1825     \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%
1826     \edef\bbl@sc@charset{\expandafter\zap@space\bbl@sc@charset\@empty}%
1827     \def\bbl@encstring##1##2{%
1828       \bbl@foreach\bbl@sc@fontenc{%
1829         \bbl@ifunset{T@####1}%
1830           {}%
1831           {\ProvideTextCommand##1{####1}{##2}%
1832            \bbl@toglobal##1%
1833            \expandafter
1834            \bbl@toglobal\csname####1\string##1\endcsname}}}%
1835     \def\bbl@sctest{%
1836       \bbl@xin@{,\bbl@opt@strings,}{,\bbl@sc@label,\bbl@sc@fontenc,}}%
1837   \fi
1838   \ifx\bbl@opt@strings\@nnil         % ie, no strings key -> defaults
1839   \else\ifx\bbl@opt@strings\relax     % ie, strings=encoded
1840     \let\AfterBabelCommands\bbl@aftercmds
1841     \let\SetString\bbl@setstring
1842     \let\bbl@stringdef\bbl@encstring
1843   \else        % ie, strings=value
1844   \bbl@sctest
1845   \ifin@
1846     \let\AfterBabelCommands\bbl@aftercmds
1847     \let\SetString\bbl@setstring
1848     \let\bbl@stringdef\bbl@provstring
1849   \fi\fi\fi
1850   \bbl@scswitch
1851   \ifx\bbl@G\@empty
1852     \def\SetString##1##2{%
1853       \bbl@error{Missing group for string \string##1}%
1854         {You must assign strings to some category, typically\\%
1855          captions or extras, but you set none}}%
1856   \fi
1857   \ifx\@empty#1%
1858     \bbl@usehooks{defaultcommands}{}%
1859   \else
1860     \@expandtwoargs
1861     \bbl@usehooks{encodedcommands}{{\bbl@sc@charset}{\bbl@sc@fontenc}}%
1862   \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) .

```
1863 \def\bbl@forlang#1#2{%
1864   \bbl@for#1\bbl@L{%
1865     \bbl@xin@{,#1,}{,\BabelLanguages,}%
1866     \ifin@#2\relax\fi}}
```

```
1867 \def\bbl@scswitch{%
1868   \bbl@forlang\bbl@tempa{%
1869     \ifx\bbl@G\@empty\else
1870       \ifx\SetString\@gobbletwo\else
1871         \edef\bbl@GL{\bbl@G\bbl@tempa}%
1872         \bbl@xin@{,\bbl@GL,}{,\bbl@screset,}%
1873         \ifin@\else
1874           \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
1875           \xdef\bbl@screset{\bbl@screset,\bbl@GL}%
1876         \fi
1877       \fi
1878     \fi}}
1879 \AtEndOfPackage{%
1880   \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{}{#2}}}%
1881   \let\bbl@scswitch\relax}
1882 \@onlypreamble\EndBabelCommands
1883 \def\EndBabelCommands{%
1884   \bbl@usehooks{stopcommands}{}%
1885   \endgroup
1886   \endgroup
1887   \bbl@scafter}
1888 \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.

```
1889 \def\bbl@setstring#1#2{% eg, \prefacename{<string>}
1890   \bbl@forlang\bbl@tempa{%
1891     \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
1892     \bbl@ifunset{\bbl@LC}% eg, \germanchaptername
1893       {\bbl@exp{%
1894         \global\\\bbl@add\<\bbl@G\bbl@tempa>{\\\bbl@scset\\#1\<\bbl@LC>}}}%
1895       {}%
1896     \def\BabelString{#2}%
1897     \bbl@usehooks{stringprocess}{}%
1898     \expandafter\bbl@stringdef
1899       \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.

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

```
1914      \fi}
1915 \else
1916   \def\bbl@scset#1#2{\def#1{#2}}
1917 \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.

```
1918 ⟨⟨∗Macros local to BabelCommands⟩⟩ ≡
1919 \def\SetStringLoop##1##2{%
1920    \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
1921    \count@\z@
1922    \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
1923      \advance\count@\@ne
1924      \toks@\expandafter{\bbl@tempa}%
1925      \bbl@exp{%
1926        \\\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
1927        \count@=\the\count@\relax}}}%
1928 ⟨⟨/Macros local to BabelCommands⟩⟩
```

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

```
1929 \def\bbl@aftercmds#1{%
1930   \toks@\expandafter{\bbl@scafter#1}%
1931   \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.

```
1932 ⟨⟨∗Macros local to BabelCommands⟩⟩ ≡
1933   \newcommand\SetCase[3][]{%
1934     \bbl@patchuclc
1935     \bbl@forlang\bbl@tempa{%
1936       \expandafter\bbl@encstring
1937         \csname\bbl@tempa @bbl@uclc\endcsname{\bbl@tempa##1}%
1938       \expandafter\bbl@encstring
1939         \csname\bbl@tempa @bbl@uc\endcsname{##2}%
1940       \expandafter\bbl@encstring
1941         \csname\bbl@tempa @bbl@lc\endcsname{##3}}}%
1942 ⟨⟨/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.

```
1943 ⟨⟨∗Macros local to BabelCommands⟩⟩ ≡
1944   \newcommand\SetHyphenMap[1]{%
1945     \bbl@forlang\bbl@tempa{%
1946       \expandafter\bbl@stringdef
1947         \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
1948 ⟨⟨/Macros local to BabelCommands⟩⟩
```

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

```
1949 \newcommand\BabelLower[2]{% one to one.
1950   \ifnum\lccode#1=#2\else
1951     \babel@savevariable{\lccode#1}%
1952     \lccode#1=#2\relax
1953   \fi}
1954 \newcommand\BabelLowerMM[4]{% many-to-many
1955   \@tempcnta=#1\relax
```

110

```
1956    \@tempcntb=#4\relax
1957    \def\bbl@tempa{%
1958      \ifnum\@tempcnta>#2\else
1959        \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
1960        \advance\@tempcnta#3\relax
1961        \advance\@tempcntb#3\relax
1962        \expandafter\bbl@tempa
1963      \fi}%
1964    \bbl@tempa}
1965  \newcommand\BabelLowerMO[4]{% many-to-one
1966    \@tempcnta=#1\relax
1967    \def\bbl@tempa{%
1968      \ifnum\@tempcnta>#2\else
1969        \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
1970        \advance\@tempcnta#3
1971        \expandafter\bbl@tempa
1972      \fi}%
1973    \bbl@tempa}
```

The following package options control the behavior of hyphenation mapping.

```
1974 ⟨⟨∗More package options⟩⟩ ≡
1975  \DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\z@}
1976  \DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap\@ne}
1977  \DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\tw@}
1978  \DeclareOption{hyphenmap=other}{\chardef\bbl@opt@hyphenmap\thr@@}
1979  \DeclareOption{hyphenmap=other*}{\chardef\bbl@opt@hyphenmap4\relax}
1980 ⟨⟨/More package options⟩⟩
```

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

```
1981  \AtEndOfPackage{%
1982    \ifx\bbl@opt@hyphenmap\@undefined
1983      \bbl@xin@{,}{\bbl@language@opts}%
1984      \chardef\bbl@opt@hyphenmap\ifin@4\else\@ne\fi
1985    \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.

```
1986  \newcommand\setlocalecaption{%  TODO. Catch typos. What about ensure?
1987    \@ifstar\bbl@setcaption@s\bbl@setcaption@x}
1988  \def\bbl@setcaption@x#1#2#3{%  language caption-name string
1989    \bbl@trim@def\bbl@tempa{#2}%
1990    \bbl@xin@{.template}{\bbl@tempa}%
1991    \ifin@
1992      \bbl@ini@captions@template{#3}{#1}%
1993    \else
1994      \edef\bbl@tempd{%
1995        \expandafter\expandafter\expandafter
1996        \strip@prefix\expandafter\meaning\csname captions#1\endcsname}%
1997      \bbl@xin@
1998        {\expandafter\string\csname #2name\endcsname}%
1999        {\bbl@tempd}%
2000      \ifin@ % Renew caption
2001        \bbl@xin@{\string\bbl@scset}{\bbl@tempd}%
2002        \ifin@
2003          \bbl@exp{%
2004            \\\bbl@ifsamestring{\bbl@tempa}{\languagename}%
2005              {\\\bbl@scset\<#2name>\<#1#2name>}%
2006              {}}%
```

```
2007        \else % Old way converts to new way
2008          \bbl@ifunset{#1#2name}%
2009            {\bbl@exp{%
2010              \\\bbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}%
2011              \\\bbl@ifsamestring{\bbl@tempa}{\languagename}%
2012                {\def\<#2name>{\<#1#2name>}}%
2013                {}}}%
2014            {}%
2015        \fi
2016      \else
2017        \bbl@xin@{\string\bbl@scset}{\bbl@tempd}% New
2018        \ifin@ % New way
2019          \bbl@exp{%
2020            \\\bbl@add\<captions#1>{\\\bbl@scset\<#2name>\<#1#2name>}%
2021            \\\bbl@ifsamestring{\bbl@tempa}{\languagename}%
2022              {\\\bbl@scset\<#2name>\<#1#2name>}%
2023              {}}%
2024        \else  % Old way, but defined in the new way
2025          \bbl@exp{%
2026            \\\bbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}%
2027            \\\bbl@ifsamestring{\bbl@tempa}{\languagename}%
2028              {\def\<#2name>{\<#1#2name>}}%
2029              {}}%
2030        \fi%
2031      \fi
2032      \@namedef{#1#2name}{#3}%
2033      \toks@\expandafter{\bbl@captionslist}%
2034      \bbl@exp{\\\in@{\<#2name>}{\the\toks@}}%
2035      \ifin@\else
2036        \bbl@exp{\\\bbl@add\\\bbl@captionslist{\<#2name>}}%
2037        \bbl@toglobal\bbl@captionslist
2038      \fi
2039    \fi}
2040 % \def\bbl@setcaption@s#1#2#3{}  % TODO. Not yet implemented
```

8.11   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.

```
2041 \bbl@trace{Macros related to glyphs}
2042 \def\set@low@box#1{\setbox\tw@\hbox{,}\setbox\z@\hbox{#1}%
2043    \dimen\z@\ht\z@ \advance\dimen\z@ -\ht\tw@%
2044    \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.

```
2045 \def\save@sf@q#1{\leavevmode
2046   \begingroup
2047     \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
2048   \endgroup}
```

8.12   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.

8.12.1   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.

```
2049 \ProvideTextCommand{\quotedblbase}{OT1}{%
2050   \save@sf@q{\set@low@box{\textquotedblright\/}%
2051     \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.

```
2052 \ProvideTextCommandDefault{\quotedblbase}{%
2053   \UseTextSymbol{OT1}{\quotedblbase}}
```

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

```
2054 \ProvideTextCommand{\quotesinglbase}{OT1}{%
2055   \save@sf@q{\set@low@box{\textquoteright\/}%
2056     \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.

```
2057 \ProvideTextCommandDefault{\quotesinglbase}{%
2058   \UseTextSymbol{OT1}{\quotesinglbase}}
```

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

```
2059 \ProvideTextCommand{\guillemetleft}{OT1}{%
2060   \ifmmode
2061     \ll
2062   \else
2063     \save@sf@q{\nobreak
2064       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2065   \fi}
2066 \ProvideTextCommand{\guillemetright}{OT1}{%
2067   \ifmmode
2068     \gg
2069   \else
2070     \save@sf@q{\nobreak
2071       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2072   \fi}
2073 \ProvideTextCommand{\guillemotleft}{OT1}{%
2074   \ifmmode
2075     \ll
2076   \else
2077     \save@sf@q{\nobreak
2078       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2079   \fi}
2080 \ProvideTextCommand{\guillemotright}{OT1}{%
2081   \ifmmode
2082     \gg
2083   \else
2084     \save@sf@q{\nobreak
2085       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2086   \fi}
```

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

```
2087 \ProvideTextCommandDefault{\guillemetleft}{%
2088   \UseTextSymbol{OT1}{\guillemetleft}}
2089 \ProvideTextCommandDefault{\guillemetright}{%
2090   \UseTextSymbol{OT1}{\guillemetright}}
2091 \ProvideTextCommandDefault{\guillemotleft}{%
2092   \UseTextSymbol{OT1}{\guillemotleft}}
2093 \ProvideTextCommandDefault{\guillemotright}{%
2094   \UseTextSymbol{OT1}{\guillemotright}}
```

The single guillemets are not available in OT1 encoding. They are faked.

```
2095 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2096   \ifmmode
2097     <%
2098   \else
2099     \save@sf@q{\nobreak
2100       \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
2101   \fi}
2102 \ProvideTextCommand{\guilsinglright}{OT1}{%
2103   \ifmmode
2104     >%
2105   \else
2106     \save@sf@q{\nobreak
2107       \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
2108   \fi}
```

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

```
2109 \ProvideTextCommandDefault{\guilsinglleft}{%
2110   \UseTextSymbol{OT1}{\guilsinglleft}}
2111 \ProvideTextCommandDefault{\guilsinglright}{%
2112   \UseTextSymbol{OT1}{\guilsinglright}}
```

8.12.2 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.

```
2113 \DeclareTextCommand{\ij}{OT1}{%
2114   i\kern-0.02em\bbl@allowhyphens j}
2115 \DeclareTextCommand{\IJ}{OT1}{%
2116   I\kern-0.02em\bbl@allowhyphens J}
2117 \DeclareTextCommand{\ij}{T1}{\char188}
2118 \DeclareTextCommand{\IJ}{T1}{\char156}
```

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

```
2119 \ProvideTextCommandDefault{\ij}{%
2120   \UseTextSymbol{OT1}{\ij}}
2121 \ProvideTextCommandDefault{\IJ}{%
2122   \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).

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

114

```
2138    \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
2139 %
2140 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2141 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}
```

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

```
2142 \ProvideTextCommandDefault{\dj}{%
2143    \UseTextSymbol{OT1}{\dj}}
2144 \ProvideTextCommandDefault{\DJ}{%
2145    \UseTextSymbol{OT1}{\DJ}}
```

\SS   For the T1 encoding \SS is defined and selects a specific glyph from the font, but for other encodings it is not available. Therefore we make it available here.

```
2146 \DeclareTextCommand{\SS}{OT1}{SS}
2147 \ProvideTextCommandDefault{\SS}{\UseTextSymbol{OT1}{\SS}}
```

8.12.3 Shorthands for quotation marks

Shorthands are provided for a number of different quotation marks, which make them usable both outside and inside mathmode. They are defined with \ProvideTextCommandDefault, but this is very likely not required because their definitions are based on encoding-dependent macros.

\glq   The 'german' single quotes.
\grq
```
2148 \ProvideTextCommandDefault{\glq}{%
2149    \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}
```

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

```
2150 \ProvideTextCommand{\grq}{T1}{%
2151    \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}
2152 \ProvideTextCommand{\grq}{TU}{%
2153    \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
2154 \ProvideTextCommand{\grq}{OT1}{%
2155    \save@sf@q{\kern-.0125em
2156       \textormath{\textquoteleft}{\mbox{\textquoteleft}}%
2157       \kern.07em\relax}}
2158 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}\grq}
```

\glqq   The 'german' double quotes.
\grqq
```
2159 \ProvideTextCommandDefault{\glqq}{%
2160    \textormath{\quotedblbase}{\mbox{\quotedblbase}}}
```

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

```
2161 \ProvideTextCommand{\grqq}{T1}{%
2162    \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2163 \ProvideTextCommand{\grqq}{TU}{%
2164    \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2165 \ProvideTextCommand{\grqq}{OT1}{%
2166    \save@sf@q{\kern-.07em
2167       \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}%
2168       \kern.07em\relax}}
2169 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}
```

\flq   The 'french' single guillemets.
\frq
```
2170 \ProvideTextCommandDefault{\flq}{%
2171    \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
2172 \ProvideTextCommandDefault{\frq}{%
2173    \textormath{\guilsinglright}{\mbox{\guilsinglright}}}
```

```
2174 \ProvideTextCommandDefault{\flqq}{%
2175   \textormath{\guillemetleft}{\mbox{\guillemetleft}}}
2176 \ProvideTextCommandDefault{\frqq}{%
2177   \textormath{\guillemetright}{\mbox{\guillemetright}}}
```

8.12.4 Umlauts and tremas

The command `\"` needs to have a different effect for different languages. For German for instance, the 'umlaut' should be positioned lower than the default position for placing it over the letters a, o, u, A, O and U. When placed over an e, i, E or I it can retain its normal position. For Dutch the same glyph is always placed in the lower position.

`\umlauthigh` To be able to provide both positions of `\"` we provide two commands to switch the positioning, the
`\umlautlow`   default will be `\umlauthigh` (the normal positioning).

```
2178 \def\umlauthigh{%
2179   \def\bbl@umlauta##1{\leavevmode\bgroup%
2180       \expandafter\accent\csname\f@encoding dqpos\endcsname
2181       ##1\bbl@allowhyphens\egroup}%
2182   \let\bbl@umlaute\bbl@umlauta}
2183 \def\umlautlow{%
2184   \def\bbl@umlauta{\protect\lower@umlaut}}
2185 \def\umlautelow{%
2186   \def\bbl@umlaute{\protect\lower@umlaut}}
2187 \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.

```
2188 \expandafter\ifx\csname U@D\endcsname\relax
2189   \csname newdimen\endcsname\U@D
2190 \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.

```
2191 \def\lower@umlaut#1{%
2192   \leavevmode\bgroup
2193     \U@D 1ex%
2194     {\setbox\z@\hbox{%
2195       \expandafter\char\csname\f@encoding dqpos\endcsname}%
2196       \dimen@ -.45ex\advance\dimen@\ht\z@
2197       \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2198     \expandafter\accent\csname\f@encoding dqpos\endcsname
2199     \fontdimen5\font\U@D #1%
2200   \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).

```
2201 \AtBeginDocument{%
```

```
2202  \DeclareTextCompositeCommand{\"}{OT1}{a}{\bbl@umlauta{a}}%
2203  \DeclareTextCompositeCommand{\"}{OT1}{e}{\bbl@umlaute{e}}%
2204  \DeclareTextCompositeCommand{\"}{OT1}{i}{\bbl@umlaute{\i}}%
2205  \DeclareTextCompositeCommand{\"}{OT1}{\i}{\bbl@umlaute{\i}}%
2206  \DeclareTextCompositeCommand{\"}{OT1}{o}{\bbl@umlauta{o}}%
2207  \DeclareTextCompositeCommand{\"}{OT1}{u}{\bbl@umlauta{u}}%
2208  \DeclareTextCompositeCommand{\"}{OT1}{A}{\bbl@umlauta{A}}%
2209  \DeclareTextCompositeCommand{\"}{OT1}{E}{\bbl@umlaute{E}}%
2210  \DeclareTextCompositeCommand{\"}{OT1}{I}{\bbl@umlaute{I}}%
2211  \DeclareTextCompositeCommand{\"}{OT1}{O}{\bbl@umlauta{O}}%
2212  \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.

```
2213 \ifx\l@english\@undefined
2214   \chardef\l@english\z@
2215 \fi
2216 % The following is used to cancel rules in ini files (see Amharic).
2217 \ifx\l@unhyphenated\@undefined
2218   \newlanguage\l@unhyphenated
2219 \fi
```

8.13 Layout

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

```
2220 \bbl@trace{Bidi layout}
2221 \providecommand\IfBabelLayout[3]{#3}%
2222 \newcommand\BabelPatchSection[1]{%
2223   \@ifundefined{#1}{}{%
2224     \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
2225     \@namedef{#1}{%
2226       \@ifstar{\bbl@presec@s{#1}}%
2227               {\@dblarg{\bbl@presec@x{#1}}}}}}
2228 \def\bbl@presec@x#1[#2]#3{%
2229   \bbl@exp{%
2230     \\\select@language@x{\bbl@main@language}%
2231     \\\bbl@cs{sspre@#1}%
2232     \\\bbl@cs{ss@#1}%
2233       [\\\foreignlanguage{\languagename}{\unexpanded{#2}}]%
2234       {\\\foreignlanguage{\languagename}{\unexpanded{#3}}}%
2235     \\\select@language@x{\languagename}}}
2236 \def\bbl@presec@s#1#2{%
2237   \bbl@exp{%
2238     \\\select@language@x{\bbl@main@language}%
2239     \\\bbl@cs{sspre@#1}%
2240     \\\bbl@cs{ss@#1}*%
2241       {\\\foreignlanguage{\languagename}{\unexpanded{#2}}}%
2242     \\\select@language@x{\languagename}}}
2243 \IfBabelLayout{sectioning}%
2244   {\BabelPatchSection{part}%
2245    \BabelPatchSection{chapter}%
2246    \BabelPatchSection{section}%
2247    \BabelPatchSection{subsection}%
2248    \BabelPatchSection{subsubsection}%
2249    \BabelPatchSection{paragraph}%
2250    \BabelPatchSection{subparagraph}%
2251    \def\babel@toc#1{%
2252      \select@language@x{\bbl@main@language}}}{}
```

```
2253 \IfBabelLayout{captions}%
2254   {\BabelPatchSection{caption}}{}
```

8.14   Load engine specific macros

```
2255 \bbl@trace{Input engine specific macros}
2256 \ifcase\bbl@engine
2257   \input txtbabel.def
2258 \or
2259   \input luababel.def
2260 \or
2261   \input xebabel.def
2262 \fi
```

8.15   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.

```
2263 \bbl@trace{Creating languages and reading ini files}
2264 \let\bbl@extend@ini\@gobble
2265 \newcommand\babelprovide[2][]{%
2266   \let\bbl@savelangname\languagename
2267   \edef\bbl@savelocaleid{\the\localeid}%
2268   % Set name and locale id
2269   \edef\languagename{#2}%
2270   \bbl@id@assign
2271   % Initialize keys
2272   \let\bbl@KVP@captions\@nil
2273   \let\bbl@KVP@date\@nil
2274   \let\bbl@KVP@import\@nil
2275   \let\bbl@KVP@main\@nil
2276   \let\bbl@KVP@script\@nil
2277   \let\bbl@KVP@language\@nil
2278   \let\bbl@KVP@hyphenrules\@nil
2279   \let\bbl@KVP@linebreaking\@nil
2280   \let\bbl@KVP@justification\@nil
2281   \let\bbl@KVP@mapfont\@nil
2282   \let\bbl@KVP@maparabic\@nil
2283   \let\bbl@KVP@mapdigits\@nil
2284   \let\bbl@KVP@intraspace\@nil
2285   \let\bbl@KVP@intrapenalty\@nil
2286   \let\bbl@KVP@onchar\@nil
2287   \let\bbl@KVP@transforms\@nil
2288   \global\let\bbl@release@transforms\@empty
2289   \let\bbl@KVP@alph\@nil
2290   \let\bbl@KVP@Alph\@nil
2291   \let\bbl@KVP@labels\@nil
2292   \bbl@csarg\let{KVP@labels*}\@nil
2293   \global\let\bbl@inidata\@empty
2294   \global\let\bbl@extend@ini\@gobble
2295   \gdef\bbl@key@list{;}%
2296   \bbl@forkv{#1}{%  TODO - error handling
2297     \in@{/}{##1}%
2298     \ifin@
2299       \global\let\bbl@extend@ini\bbl@extend@ini@aux
2300       \bbl@renewinikey##1\@@{##2}%
2301     \else
2302       \bbl@csarg\def{KVP@##1}{##2}%
```

```
2303    \fi}%
2304 \chardef\bbl@howloaded=% 0:none; 1:ldf without ini; 2:ini
2305    \bbl@ifunset{date#2}\z@{\bbl@ifunset{bbl@llevel@#2}\@ne\tw@}%
2306 % == init ==
2307 \ifx\bbl@screset\@undefined
2308    \bbl@ldfinit
2309 \fi
2310 % ==
2311 \let\bbl@lbkflag\relax % \@empty = do setup linebreak
2312 \ifcase\bbl@howloaded
2313    \let\bbl@lbkflag\@empty % new
2314 \else
2315    \ifx\bbl@KVP@hyphenrules\@nil\else
2316        \let\bbl@lbkflag\@empty
2317    \fi
2318    \ifx\bbl@KVP@import\@nil\else
2319        \let\bbl@lbkflag\@empty
2320    \fi
2321 \fi
2322 % == import, captions ==
2323 \ifx\bbl@KVP@import\@nil\else
2324    \bbl@exp{\\\bbl@ifblank{\bbl@KVP@import}}%
2325        {\ifx\bbl@initoload\relax
2326            \begingroup
2327                \def\BabelBeforeIni##1##2{\gdef\bbl@KVP@import{##1}\endinput}%
2328                \bbl@input@texini{#2}%
2329            \endgroup
2330         \else
2331            \xdef\bbl@KVP@import{\bbl@initoload}%
2332         \fi}%
2333        {}%
2334 \fi
2335 \ifx\bbl@KVP@captions\@nil
2336    \let\bbl@KVP@captions\bbl@KVP@import
2337 \fi
2338 % ==
2339 \ifx\bbl@KVP@transforms\@nil\else
2340    \bbl@replace\bbl@KVP@transforms{ }{,}%
2341 \fi
2342 % == Load ini ==
2343 \ifcase\bbl@howloaded
2344    \bbl@provide@new{#2}%
2345 \else
2346    \bbl@ifblank{#1}%
2347        {}%  With \bbl@load@basic below
2348        {\bbl@provide@renew{#2}}%
2349 \fi
2350 % Post tasks
2351 % ----------
2352 % == subsequent calls after the first provide for a locale ==
2353 \ifx\bbl@inidata\@empty\else
2354    \bbl@extend@ini{#2}%
2355 \fi
2356 % == ensure captions ==
2357 \ifx\bbl@KVP@captions\@nil\else
2358    \bbl@ifunset{bbl@extracaps@#2}%
2359        {\bbl@exp{\\\babelensure[exclude=\\\today]{#2}}}%
2360        {\bbl@exp{\\\babelensure[exclude=\\\today,
2361                include=\[bbl@extracaps@#2]]{#2}}}%
```

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

```
2421        end}%
2422      \ifx\bbl@mapselect\@undefined  % TODO. almost the same as mapfont
2423        \AtBeginDocument{%
2424          \bbl@patchfont{{\bbl@mapselect}}%
2425          {\selectfont}}%
2426        \def\bbl@mapselect{%
2427          \let\bbl@mapselect\relax
2428          \edef\bbl@prefontid{\fontid\font}}%
2429        \def\bbl@mapdir##1{%
2430          {\def\languagename{##1}%
2431           \let\bbl@ifrestoring\@firstoftwo % To avoid font warning
2432           \bbl@switchfont
2433           \ifnum\fontid\font>\z@ % A hack, for the pgf nullfont hack
2434             \directlua{
2435               Babel.locale_props[\the\csname bbl@id@@##1\endcsname]%
2436                         ['/\bbl@prefontid'] = \fontid\font\space}%
2437           \fi}}%
2438      \fi
2439      \bbl@exp{\\\bbl@add\\\bbl@mapselect{\\\bbl@mapdir{\languagename}}}%
2440    \fi
2441    % TODO - catch non-valid values
2442  \fi
2443  % == mapfont ==
2444  % For bidi texts, to switch the font based on direction
2445  \ifx\bbl@KVP@mapfont\@nil\else
2446    \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}{}%
2447      {\bbl@error{Option '\bbl@KVP@mapfont' unknown for\\%
2448                  mapfont. Use 'direction'.%
2449                  {See the manual for details.}}}%
2450    \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
2451    \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
2452    \ifx\bbl@mapselect\@undefined % TODO. See onchar.
2453      \AtBeginDocument{%
2454        \bbl@patchfont{{\bbl@mapselect}}%
2455        {\selectfont}}%
2456      \def\bbl@mapselect{%
2457        \let\bbl@mapselect\relax
2458        \edef\bbl@prefontid{\fontid\font}}%
2459      \def\bbl@mapdir##1{%
2460        {\def\languagename{##1}%
2461         \let\bbl@ifrestoring\@firstoftwo % avoid font warning
2462         \bbl@switchfont
2463         \directlua{Babel.fontmap
2464           [\the\csname bbl@wdir@##1\endcsname]%
2465           [\bbl@prefontid]=\fontid\font}}}%
2466    \fi
2467    \bbl@exp{\\\bbl@add\\\bbl@mapselect{\\\bbl@mapdir{\languagename}}}%
2468  \fi
2469  % == Line breaking: intraspace, intrapenalty ==
2470  % For CJK, East Asian, Southeast Asian, if interspace in ini
2471  \ifx\bbl@KVP@intraspace\@nil\else % We can override the ini or set
2472    \bbl@csarg\edef{intsp@#2}{\bbl@KVP@intraspace}%
2473  \fi
2474  \bbl@provide@intraspace
2475  % == Line breaking: CJK quotes ==
2476  \ifcase\bbl@engine\or
2477    \bbl@xin@{/c}{/\bbl@cl{lnbrk}}%
2478    \ifin@
2479      \bbl@ifunset{bbl@quote@\languagename}{}%
```

121

```
2480        {\directlua{
2481          Babel.locale_props[\the\localeid].cjk_quotes = {}
2482          local cs = 'op'
2483          for c in string.utfvalues(%
2484            [[\csname bbl@quote@\languagename\endcsname]]) do
2485            if Babel.cjk_characters[c].c == 'qu' then
2486              Babel.locale_props[\the\localeid].cjk_quotes[c] = cs
2487            end
2488            cs = ( cs == 'op') and 'cl' or 'op'
2489          end
2490        }}%
2491    \fi
2492  \fi
2493  % == Line breaking: justification ==
2494  \ifx\bbl@KVP@justification\@nil\else
2495    \let\bbl@KVP@linebreaking\bbl@KVP@justification
2496  \fi
2497  \ifx\bbl@KVP@linebreaking\@nil\else
2498    \bbl@xin@{,\bbl@KVP@linebreaking,}{,elongated,kashida,cjk,unhyphenated,}%
2499    \ifin@
2500      \bbl@csarg\xdef
2501        {lnbrk@\languagename}{\expandafter\@car\bbl@KVP@linebreaking\@nil}%
2502    \fi
2503  \fi
2504  \bbl@xin@{/e}{/\bbl@cl{lnbrk}}%
2505  \ifin@\else\bbl@xin@{/k}{/\bbl@cl{lnbrk}}\fi
2506  \ifin@\bbl@arabicjust\fi
2507  % == Line breaking: hyphenate.other.(locale|script) ==
2508  \ifx\bbl@lbkflag\@empty
2509    \bbl@ifunset{bbl@hyotl@\languagename}{}%
2510      {\bbl@csarg\bbl@replace{hyotl@\languagename}{ }{,}%
2511      \bbl@startcommands*{\languagename}{}%
2512        \bbl@csarg\bbl@foreach{hyotl@\languagename}{%
2513          \ifcase\bbl@engine
2514            \ifnum##1<257
2515              \SetHyphenMap{\BabelLower{##1}{##1}}%
2516            \fi
2517          \else
2518            \SetHyphenMap{\BabelLower{##1}{##1}}%
2519          \fi}%
2520      \bbl@endcommands}%
2521    \bbl@ifunset{bbl@hyots@\languagename}{}%
2522      {\bbl@csarg\bbl@replace{hyots@\languagename}{ }{,}%
2523      \bbl@csarg\bbl@foreach{hyots@\languagename}{%
2524        \ifcase\bbl@engine
2525          \ifnum##1<257
2526            \global\lccode##1=##1\relax
2527          \fi
2528        \else
2529          \global\lccode##1=##1\relax
2530        \fi}}%
2531  \fi
2532  % == Counters: maparabic ==
2533  % Native digits, if provided in ini (TeX level, xe and lua)
2534  \ifcase\bbl@engine\else
2535    \bbl@ifunset{bbl@dgnat@\languagename}{}%
2536      {\expandafter\ifx\csname bbl@dgnat@\languagename\endcsname\@empty\else
2537        \expandafter\expandafter\expandafter
2538        \bbl@setdigits\csname bbl@dgnat@\languagename\endcsname
```

```
2539          \ifx\bbl@KVP@maparabic\@nil\else
2540            \ifx\bbl@latinarabic\@undefined
2541              \expandafter\let\expandafter\@arabic
2542                \csname bbl@counter@\languagename\endcsname
2543            \else    % ie, if layout=counters, which redefines \@arabic
2544              \expandafter\let\expandafter\bbl@latinarabic
2545                \csname bbl@counter@\languagename\endcsname
2546            \fi
2547          \fi
2548        \fi}%
2549    \fi
2550    % == Counters: mapdigits ==
2551    % Native digits (lua level).
2552    \ifodd\bbl@engine
2553      \ifx\bbl@KVP@mapdigits\@nil\else
2554        \bbl@ifunset{bbl@dgnat@\languagename}{}%
2555          {\RequirePackage{luatexbase}%
2556           \bbl@activate@preotf
2557           \directlua{
2558             Babel = Babel or {}  %%% -> presets in luababel
2559             Babel.digits_mapped = true
2560             Babel.digits = Babel.digits or {}
2561             Babel.digits[\the\localeid] =
2562               table.pack(string.utfvalue('\bbl@cl{dgnat}'))
2563             if not Babel.numbers then
2564               function Babel.numbers(head)
2565                 local LOCALE = Babel.attr_locale
2566                 local GLYPH = node.id'glyph'
2567                 local inmath = false
2568                 for item in node.traverse(head) do
2569                   if not inmath and item.id == GLYPH then
2570                     local temp = node.get_attribute(item, LOCALE)
2571                     if Babel.digits[temp] then
2572                       local chr = item.char
2573                       if chr > 47 and chr < 58 then
2574                         item.char = Babel.digits[temp][chr-47]
2575                       end
2576                     end
2577                   elseif item.id == node.id'math' then
2578                     inmath = (item.subtype == 0)
2579                   end
2580                 end
2581                 return head
2582               end
2583             end
2584          }}%
2585      \fi
2586    \fi
2587    % == Counters: alph, Alph ==
2588    % What if extras<lang> contains a \babel@save\@alph? It won't be
2589    % restored correctly when exiting the language, so we ignore
2590    % this change with the \bbl@alph@saved trick.
2591    \ifx\bbl@KVP@alph\@nil\else
2592      \bbl@extras@wrap{\\\bbl@alph@saved}%
2593        {\let\bbl@alph@saved\@alph}%
2594        {\let\@alph\bbl@alph@saved
2595         \babel@save\@alph}%
2596      \bbl@exp{%
2597        \\\bbl@add\<extras\languagename>{%
```

```
2598          \let\\\@alph\<bbl@cntr@\bbl@KVP@alph @\languagename>}}%
2599   \fi
2600   \ifx\bbl@KVP@Alph\@nil\else
2601     \bbl@extras@wrap{\\\bbl@Alph@saved}%
2602        {\let\bbl@Alph@saved\@Alph}%
2603        {\let\@Alph\bbl@Alph@saved
2604         \babel@save\@Alph}%
2605     \bbl@exp{%
2606        \\\bbl@add\<extras\languagename>{%
2607          \let\\\@Alph\<bbl@cntr@\bbl@KVP@Alph @\languagename>}}%
2608   \fi
2609   % == require.babel in ini ==
2610   % To load or reaload the babel-*.tex, if require.babel in ini
2611   \ifx\bbl@beforestart\relax\else  % But not in doc aux or body
2612     \bbl@ifunset{bbl@rqtex@\languagename}{}%
2613        {\expandafter\ifx\csname bbl@rqtex@\languagename\endcsname\@empty\else
2614          \let\BabelBeforeIni\@gobbletwo
2615          \chardef\atcatcode=\catcode`\@
2616          \catcode`\@=11\relax
2617          \bbl@input@texini{\bbl@cs{rqtex@\languagename}}%
2618          \catcode`\@=\atcatcode
2619          \let\atcatcode\relax
2620          \global\bbl@csarg\let{rqtex@\languagename}\relax
2621        \fi}%
2622   \fi
2623   % == frenchspacing ==
2624   \ifcase\bbl@howloaded\in@true\else\in@false\fi
2625   \ifin@\else\bbl@xin@{typography/frenchspacing}{\bbl@key@list}\fi
2626   \ifin@
2627     \bbl@extras@wrap{\\\bbl@pre@fs}%
2628        {\bbl@pre@fs}%
2629        {\bbl@post@fs}%
2630   \fi
2631   % == Release saved transforms ==
2632   \bbl@release@transforms\relax % \relax closes the last item.
2633   % == main ==
2634   \ifx\bbl@KVP@main\@nil  % Restore only if not 'main'
2635     \let\languagename\bbl@savelangname
2636     \chardef\localeid\bbl@savelocaleid\relax
2637   \fi}
```

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

```
2638 \def\bbl@provide@new#1{%
2639   \@namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
2640   \@namedef{extras#1}{}%
2641   \@namedef{noextras#1}{}%
2642   \bbl@startcommands*{#1}{captions}%
2643     \ifx\bbl@KVP@captions\@nil %        and also if import, implicit
2644        \def\bbl@tempb##1{%                elt for \bbl@captionslist
2645          \ifx##1\@empty\else
2646            \bbl@exp{%
2647              \\\SetString\\##1{%
2648                \\\bbl@nocaption{\bbl@stripslash##1}{#1\bbl@stripslash##1}}}%
2649            \expandafter\bbl@tempb
2650          \fi}%
2651        \expandafter\bbl@tempb\bbl@captionslist\@empty
2652     \else
2653        \ifx\bbl@initoload\relax
```

```
2654        \bbl@read@ini{\bbl@KVP@captions}2%  % Here letters cat = 11
2655      \else
2656        \bbl@read@ini{\bbl@initoload}2%     % Same
2657      \fi
2658    \fi
2659  \StartBabelCommands*{#1}{date}%
2660    \ifx\bbl@KVP@import\@nil
2661      \bbl@exp{%
2662        \\\SetString\\\today{\\\bbl@nocaption{today}{#1today}}}%
2663    \else
2664      \bbl@savetoday
2665      \bbl@savedate
2666    \fi
2667  \bbl@endcommands
2668  \bbl@load@basic{#1}%
2669  % == hyphenmins == (only if new)
2670  \bbl@exp{%
2671    \gdef\<#1hyphenmins>{%
2672      {\bbl@ifunset{bbl@lfthm@#1}{2}{\bbl@cs{lfthm@#1}}}%
2673      {\bbl@ifunset{bbl@rgthm@#1}{3}{\bbl@cs{rgthm@#1}}}}}%
2674  % == hyphenrules (also in renew) ==
2675  \bbl@provide@hyphens{#1}%
2676  \ifx\bbl@KVP@main\@nil\else
2677    \expandafter\main@language\expandafter{#1}%
2678  \fi}
2679 %
2680 \def\bbl@provide@renew#1{%
2681  \ifx\bbl@KVP@captions\@nil\else
2682    \StartBabelCommands*{#1}{captions}%
2683      \bbl@read@ini{\bbl@KVP@captions}2%   % Here all letters cat = 11
2684    \EndBabelCommands
2685  \fi
2686  \ifx\bbl@KVP@import\@nil\else
2687    \StartBabelCommands*{#1}{date}%
2688      \bbl@savetoday
2689      \bbl@savedate
2690    \EndBabelCommands
2691  \fi
2692  % == hyphenrules (also in new) ==
2693  \ifx\bbl@lbkflag\@empty
2694    \bbl@provide@hyphens{#1}%
2695  \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.)

```
2696 \def\bbl@load@basic#1{%
2697  \ifcase\bbl@howloaded\or\or
2698    \ifcase\csname bbl@llevel@\languagename\endcsname
2699      \bbl@csarg\let{lname@\languagename}\relax
2700    \fi
2701  \fi
2702  \bbl@ifunset{bbl@lname@#1}%
2703    {\def\BabelBeforeIni##1##2{%
2704       \begingroup
2705         \let\bbl@ini@captions@aux\@gobbletwo
2706         \def\bbl@inidate ####1.####2.####3.####4\relax ####5####6{}%
2707         \bbl@read@ini{##1}1%
2708         \ifx\bbl@initoload\relax\endinput\fi
```

125

```
2709        \endgroup}%
2710     \begingroup         % boxed, to avoid extra spaces:
2711       \ifx\bbl@initoload\relax
2712         \bbl@input@texini{#1}%
2713       \else
2714         \setbox\z@\hbox{\BabelBeforeIni{\bbl@initoload}{}}%
2715       \fi
2716     \endgroup}%
2717    {}}
```

The hyphenrules option is handled with an auxiliary macro.

```
2718 \def\bbl@provide@hyphens#1{%
2719   \let\bbl@tempa\relax
2720   \ifx\bbl@KVP@hyphenrules\@nil\else
2721     \bbl@replace\bbl@KVP@hyphenrules{ }{,}%
2722     \bbl@foreach\bbl@KVP@hyphenrules{%
2723       \ifx\bbl@tempa\relax    % if not yet found
2724         \bbl@ifsamestring{##1}{+}%
2725           {{\bbl@exp{\\\addlanguage\<l@##1>}}}%
2726           {}%
2727         \bbl@ifunset{l@##1}%
2728           {}%
2729           {\bbl@exp{\let\bbl@tempa\<l@##1>}}%
2730       \fi}%
2731   \fi
2732   \ifx\bbl@tempa\relax %         if no opt or no language in opt found
2733     \ifx\bbl@KVP@import\@nil
2734       \ifx\bbl@initoload\relax\else
2735         \bbl@exp{%              and hyphenrules is not empty
2736           \\\bbl@ifblank{\bbl@cs{hyphr@#1}}%
2737             {}%
2738             {\let\\\bbl@tempa\<l@\bbl@cl{hyphr}>}}%
2739       \fi
2740     \else % if importing
2741       \bbl@exp{%                 and hyphenrules is not empty
2742         \\\bbl@ifblank{\bbl@cs{hyphr@#1}}%
2743           {}%
2744           {\let\\\bbl@tempa\<l@\bbl@cl{hyphr}>}}%
2745     \fi
2746   \fi
2747   \bbl@ifunset{bbl@tempa}%       ie, relax or undefined
2748     {\bbl@ifunset{l@#1}%          no hyphenrules found - fallback
2749       {\bbl@exp{\\\adddialect\<l@#1>\language}}%
2750       {}}%                        so, l@<lang> is ok - nothing to do
2751     {\bbl@exp{\\\adddialect\<l@#1>\bbl@tempa}}}% found in opt list or ini
```

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

```
2752 \def\bbl@input@texini#1{%
2753   \bbl@bsphack
2754     \bbl@exp{%
2755       \catcode`\\\%=14 \catcode`\\\\=0
2756       \catcode`\\\{=1  \catcode`\\\}=2
2757       \lowercase{\\\InputIfFileExists{babel-#1.tex}{}{}}}%
2758     \catcode`\\\%=\the\catcode`\%\relax
2759     \catcode`\\\\=\the\catcode`\\\relax
2760     \catcode`\\\{=\the\catcode`\{\relax
2761     \catcode`\\\}=\the\catcode`\}\relax}%
2762   \bbl@esphack
```

126

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.

```
2763 \def\bbl@iniline#1\bbl@iniline{%
2764   \@ifnextchar[\bbl@inisect{\@ifnextchar;\bbl@iniskip\bbl@inistore}#1\@@}% ]
2765 \def\bbl@inisect[#1]#2\@@{\def\bbl@section{#1}}
2766 \def\bbl@iniskip#1\@@{}%       if starts with ;
2767 \def\bbl@inistore#1=#2\@@{%       full (default)
2768   \bbl@trim@def\bbl@tempa{#1}%
2769   \bbl@trim\toks@{#2}%
2770   \bbl@xin@{;\bbl@section/\bbl@tempa;}{\bbl@key@list}%
2771   \ifin@\else
2772     \bbl@exp{%
2773       \\\g@addto@macro\\\bbl@inidata{%
2774         \\\bbl@elt{\bbl@section}{\bbl@tempa}{\the\toks@}}}%
2775   \fi}
2776 \def\bbl@inistore@min#1=#2\@@{%  minimal (maybe set in \bbl@read@ini)
2777   \bbl@trim@def\bbl@tempa{#1}%
2778   \bbl@trim\toks@{#2}%
2779   \bbl@xin@{.identification.}{.\bbl@section.}%
2780   \ifin@
2781     \bbl@exp{\\\g@addto@macro\\\bbl@inidata{%
2782       \\\bbl@elt{identification}{\bbl@tempa}{\the\toks@}}}%
2783   \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.

```
2784 \ifx\bbl@readstream\@undefined
2785   \csname newread\endcsname\bbl@readstream
2786 \fi
2787 \def\bbl@read@ini#1#2{%
2788   \global\let\bbl@extend@ini\@gobble
2789   \openin\bbl@readstream=babel-#1.ini
2790   \ifeof\bbl@readstream
2791     \bbl@error
2792       {There is no ini file for the requested language\\%
2793        (#1). Perhaps you misspelled it or your installation\\%
2794        is not complete.}%
2795       {Fix the name or reinstall babel.}%
2796   \else
2797     % == Store ini data in \bbl@inidata ==
2798     \catcode`\[=12 \catcode`\]=12 \catcode`\==12 \catcode`\&=12
2799     \catcode`\;=12 \catcode`\|=12 \catcode`\%=14 \catcode`\-=12
2800     \bbl@info{Importing
2801              \ifcase#2font and identification \or basic \fi
2802                data for \languagename\\%
2803            from babel-#1.ini. Reported}%
2804     \ifnum#2=\z@
2805       \global\let\bbl@inidata\@empty
2806       \let\bbl@inistore\bbl@inistore@min    % Remember it's local
2807     \fi
2808     \def\bbl@section{identification}%
2809     \bbl@exp{\\\bbl@inistore tag.ini=#1\\\@@}%
2810     \bbl@inistore load.level=#2\@@
2811     \loop
```

127

```
2812    \if T\ifeof\bbl@readstream F\fi T\relax % Trick, because inside \loop
2813      \endlinechar\m@ne
2814      \read\bbl@readstream to \bbl@line
2815      \endlinechar`\^^M
2816      \ifx\bbl@line\@empty\else
2817        \expandafter\bbl@iniline\bbl@line\bbl@iniline
2818      \fi
2819    \repeat
2820    % == Process stored data ==
2821    \bbl@csarg\xdef{lini@\languagename}{#1}%
2822    \bbl@read@ini@aux
2823    % == 'Export' data ==
2824    \bbl@ini@exports{#2}%
2825    \global\bbl@csarg\let{inidata@\languagename}\bbl@inidata
2826    \global\let\bbl@inidata\@empty
2827    \bbl@exp{\\\bbl@add@list\\\bbl@ini@loaded{\languagename}}%
2828    \bbl@toglobal\bbl@ini@loaded
2829  \fi}
2830 \def\bbl@read@ini@aux{%
2831    \let\bbl@savestrings\@empty
2832    \let\bbl@savetoday\@empty
2833    \let\bbl@savedate\@empty
2834    \def\bbl@elt##1##2##3{%
2835      \def\bbl@section{##1}%
2836      \in@{=date.}{=##1}% Find a better place
2837      \ifin@
2838        \bbl@ini@calendar{##1}%
2839      \fi
2840      \bbl@ifunset{bbl@inikv@##1}{}%
2841        {\csname bbl@inikv@##1\endcsname{##2}{##3}}}%
2842    \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.

```
2843 \def\bbl@extend@ini@aux#1{%
2844    \bbl@startcommands*{#1}{captions}%
2845      % Activate captions/... and modify exports
2846    \bbl@csarg\def{inikv@captions.licr}##1##2{%
2847        \setlocalecaption{#1}{##1}{##2}}%
2848    \def\bbl@inikv@captions##1##2{%
2849        \bbl@ini@captions@aux{##1}{##2}}%
2850    \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2851    \def\bbl@exportkey##1##2##3{%
2852      \bbl@ifunset{bbl@@kv@##2}{}%
2853        {\expandafter\ifx\csname bbl@@kv@##2\endcsname\@empty\else
2854            \bbl@exp{\global\let\<bbl@##1@\languagename>\<bbl@@kv@##2>}%
2855        \fi}}%
2856      % As with \bbl@read@ini, but with some changes
2857    \bbl@read@ini@aux
2858    \bbl@ini@exports\tw@
2859      % Update inidata@lang by pretending the ini is read.
2860    \def\bbl@elt##1##2##3{%
2861      \def\bbl@section{##1}%
2862      \bbl@iniline##2=##3\bbl@iniline}%
2863    \csname bbl@inidata@#1\endcsname
2864    \global\bbl@csarg\let{inidata@#1}\bbl@inidata
2865    \StartBabelCommands*{#1}{date}% And from the import stuff
2866    \def\bbl@stringdef##1##2{\gdef##1{##2}}%
2867    \bbl@savetoday
```

128

```
2868      \bbl@savedate
2869   \bbl@endcommands}
```

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

```
2870 \def\bbl@ini@calendar#1{%
2871 \lowercase{\def\bbl@tempa{=#1=}}%
2872 \bbl@replace\bbl@tempa{=date.gregorian}{}%
2873 \bbl@replace\bbl@tempa{=date.}{}%
2874 \in@{.licr=}{#1=}%
2875 \ifin@
2876   \ifcase\bbl@engine
2877     \bbl@replace\bbl@tempa{.licr=}{}%
2878   \else
2879     \let\bbl@tempa\relax
2880   \fi
2881 \fi
2882 \ifx\bbl@tempa\relax\else
2883   \bbl@replace\bbl@tempa{=}{}%
2884   \bbl@exp{%
2885     \def\<bbl@inikv@#1>####1####2{%
2886       \\\bbl@inidate####1...\relax{####2}{\bbl@tempa}}}%
2887 \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).

```
2888 \def\bbl@renewinikey#1/#2\@@#3{%
2889   \edef\bbl@tempa{\zap@space #1 \@empty}%    section
2890   \edef\bbl@tempb{\zap@space #2 \@empty}%    key
2891   \bbl@trim\toks@{#3}%                        value
2892   \bbl@exp{%
2893     \edef\\\bbl@key@list{\bbl@key@list \bbl@tempa/\bbl@tempb;}%
2894     \\\g@addto@macro\\\bbl@inidata{%
2895       \\\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.

```
2896 \def\bbl@exportkey#1#2#3{%
2897   \bbl@ifunset{bbl@@kv@#2}%
2898     {\bbl@csarg\gdef{#1@\languagename}{#3}}%
2899     {\expandafter\ifx\csname bbl@@kv@#2\endcsname\@empty
2900       \bbl@csarg\gdef{#1@\languagename}{#3}%
2901     \else
2902       \bbl@exp{\global\let\<bbl@#1@\languagename>\<bbl@@kv@#2>}%
2903     \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.

```
2904 \def\bbl@iniwarning#1{%
2905   \bbl@ifunset{bbl@@kv@identification.warning#1}{}%
2906     {\bbl@warning{%
2907       From babel-\bbl@cs{lini@\languagename}.ini:\\%
2908       \bbl@cs{@kv@identification.warning#1}\\%
2909       Reported }}}
2910 %
2911 \let\bbl@release@transforms\@empty
2912 %
```

129

```
2913 \def\bbl@ini@exports#1{%
2914   % Identification always exported
2915   \bbl@iniwarning{}%
2916   \ifcase\bbl@engine
2917     \bbl@iniwarning{.pdflatex}%
2918   \or
2919     \bbl@iniwarning{.lualatex}%
2920   \or
2921     \bbl@iniwarning{.xelatex}%
2922   \fi%
2923   \bbl@exportkey{llevel}{identification.load.level}{}%
2924   \bbl@exportkey{elname}{identification.name.english}{}%
2925   \bbl@exp{\\\bbl@exportkey{lname}{identification.name.opentype}%
2926     {\csname bbl@elname@\languagename\endcsname}}%
2927   \bbl@exportkey{tbcp}{identification.tag.bcp47}{}%
2928   \bbl@exportkey{lbcp}{identification.language.tag.bcp47}{}%
2929   \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
2930   \bbl@exportkey{esname}{identification.script.name}{}%
2931   \bbl@exp{\\\bbl@exportkey{sname}{identification.script.name.opentype}%
2932     {\csname bbl@esname@\languagename\endcsname}}%
2933   \bbl@exportkey{sbcp}{identification.script.tag.bcp47}{}%
2934   \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
2935   % Also maps bcp47 -> languagename
2936   \ifbbl@bcptoname
2937     \bbl@csarg\xdef{bcp@map@\bbl@cl{tbcp}}{\languagename}%
2938   \fi
2939   % Conditional
2940   \ifnum#1>\z@          % 0 = only info, 1, 2 = basic, (re)new
2941     \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
2942     \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
2943     \bbl@exportkey{lfthm}{typography.lefthyphenmin}{2}%
2944     \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
2945     \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
2946     \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
2947     \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
2948     \bbl@exportkey{intsp}{typography.intraspace}{}%
2949     \bbl@exportkey{frspc}{typography.frenchspacing}{u}%
2950     \bbl@exportkey{chrng}{characters.ranges}{}%
2951     \bbl@exportkey{quote}{characters.delimiters.quotes}{}%
2952     \bbl@exportkey{dgnat}{numbers.digits.native}{}%
2953     \ifnum#1=\tw@          % only (re)new
2954       \bbl@exportkey{rqtex}{identification.require.babel}{}%
2955       \bbl@toglobal\bbl@savetoday
2956       \bbl@toglobal\bbl@savedate
2957       \bbl@savestrings
2958     \fi
2959   \fi}
```

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

```
2960 \def\bbl@inikv#1#2{%       key=value
2961   \toks@{#2}%              This hides #'s from ini values
2962   \bbl@csarg\edef{@kv@\bbl@section.#1}{\the\toks@}}
```

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

```
2963 \let\bbl@inikv@identification\bbl@inikv
2964 \let\bbl@inikv@typography\bbl@inikv
2965 \let\bbl@inikv@characters\bbl@inikv
2966 \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'.

```
2967 \def\bbl@inikv@counters#1#2{%
2968   \bbl@ifsamestring{#1}{digits}%
2969     {\bbl@error{The counter name 'digits' is reserved for mapping\\%
2970                 decimal digits}%
2971                {Use another name.}}%
2972     {}%
2973   \def\bbl@tempc{#1}%
2974   \bbl@trim@def{\bbl@tempb*}{#2}%
2975   \in@{.1$}{#1$}%
2976   \ifin@
2977     \bbl@replace\bbl@tempc{.1}{}%
2978     \bbl@csarg\protected@xdef{cntr@\bbl@tempc @\languagename}{%
2979       \noexpand\bbl@alphnumeral{\bbl@tempc}}%
2980   \fi
2981   \in@{.F.}{#1}%
2982   \ifin@\else\in@{.S.}{#1}\fi
2983   \ifin@
2984     \bbl@csarg\protected@xdef{cntr@#1@\languagename}{\bbl@tempb*}%
2985   \else
2986     \toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa
2987     \expandafter\bbl@buildifcase\bbl@tempb* \\ % Space after \\
2988     \bbl@csarg{\global\expandafter\let}{cntr@#1@\languagename}\bbl@tempa
2989   \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.

```
2990 \ifcase\bbl@engine
2991   \bbl@csarg\def{inikv@captions.licr}#1#2{%
2992     \bbl@ini@captions@aux{#1}{#2}}
2993 \else
2994   \def\bbl@inikv@captions#1#2{%
2995     \bbl@ini@captions@aux{#1}{#2}}
2996 \fi
```

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

```
2997 \def\bbl@ini@captions@template#1#2{% string language tempa=capt-name
2998   \bbl@replace\bbl@tempa{.template}{}%
2999   \def\bbl@toreplace{#1{}}%
3000   \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
3001   \bbl@replace\bbl@toreplace{[[}{\csname}%
3002   \bbl@replace\bbl@toreplace{[}{\csname the}%
3003   \bbl@replace\bbl@toreplace{]]}{name\endcsname{}}%
3004   \bbl@replace\bbl@toreplace{]}{\endcsname{}}%
3005   \bbl@xin@{,\bbl@tempa,}{,chapter,appendix,part,}%
3006   \ifin@
3007     \@nameuse{bbl@patch\bbl@tempa}%
3008     \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
3009   \fi
3010   \bbl@xin@{,\bbl@tempa,}{,figure,table,}%
3011   \ifin@
3012     \toks@\expandafter{\bbl@toreplace}%
3013     \bbl@exp{\gdef\<fnum@\bbl@tempa>{\the\toks@}}%
3014   \fi}
3015 \def\bbl@ini@captions@aux#1#2{%
3016   \bbl@trim@def\bbl@tempa{#1}%
```

```
3017    \bbl@xin@{.template}{\bbl@tempa}%
3018    \ifin@
3019      \bbl@ini@captions@template{#2}\languagename
3020    \else
3021      \bbl@ifblank{#2}%
3022        {\bbl@exp{%
3023          \toks@{\\\bbl@nocaption{\bbl@tempa}{\languagename\bbl@tempa name}}}}%
3024        {\bbl@trim\toks@{#2}}%
3025      \bbl@exp{%
3026        \\\bbl@add\\\bbl@savestrings{%
3027          \\\SetString\<\bbl@tempa name>{\the\toks@}}}%
3028      \toks@\expandafter{\bbl@captionslist}%
3029      \bbl@exp{\\\in@{\<\bbl@tempa name>}{\the\toks@}}%
3030      \ifin@\else
3031        \bbl@exp{%
3032          \\\bbl@add\<bbl@extracaps@\languagename>{\<\bbl@tempa name>}%
3033          \\\bbl@toglobal\<bbl@extracaps@\languagename>}%
3034      \fi
3035    \fi}
```

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

```
3036 \def\bbl@list@the{%
3037   part,chapter,section,subsection,subsubsection,paragraph,%
3038   subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,%
3039   table,page,footnote,mpfootnote,mpfn}
3040 \def\bbl@map@cnt#1{%  #1:roman,etc, // #2:enumi,etc
3041   \bbl@ifunset{bbl@map@#1@\languagename}%
3042     {\@nameuse{#1}}%
3043     {\@nameuse{bbl@map@#1@\languagename}}}
3044 \def\bbl@inikv@labels#1#2{%
3045   \in@{.map}{#1}%
3046   \ifin@
3047     \ifx\bbl@KVP@labels\@nil\else
3048       \bbl@xin@{ map }{ \bbl@KVP@labels\space}%
3049       \ifin@
3050         \def\bbl@tempc{#1}%
3051         \bbl@replace\bbl@tempc{.map}{}%
3052         \in@{,#2,}{,arabic,roman,Roman,alph,Alph,fnsymbol,}%
3053         \bbl@exp{%
3054           \gdef\<bbl@map@\bbl@tempc @\languagename>%
3055             {\ifin@\<#2>\else\\\localcounter{#2}\fi}}%
3056         \bbl@foreach\bbl@list@the{%
3057           \bbl@ifunset{the##1}{}%
3058             {\bbl@exp{\let\\\bbl@tempd\<the##1>}%
3059              \bbl@exp{%
3060                \\\bbl@sreplace\<the##1>%
3061                  {\<\bbl@tempc>{##1}}{\\\bbl@map@cnt{\bbl@tempc}{##1}}%
3062                \\\bbl@sreplace\<the##1>%
3063                  {\<\@empty @\bbl@tempc>\<c@##1>}{\\\bbl@map@cnt{\bbl@tempc}{##1}}}%
3064              \expandafter\ifx\csname the##1\endcsname\bbl@tempd\else
3065                \toks@\expandafter\expandafter\expandafter{%
3066                  \csname the##1\endcsname}%
3067                \expandafter\xdef\csname the##1\endcsname{{\the\toks@}}%
3068              \fi}}%
3069       \fi
3070     \fi
3071   %
3072   \else
3073     %
```

132

```
3074    % The following code is still under study. You can test it and make
3075    % suggestions. Eg, enumerate.2 = ([enumi]).([enumii]). It's
3076    % language dependent.
3077    \in@{enumerate.}{#1}%
3078    \ifin@
3079      \def\bbl@tempa{#1}%
3080      \bbl@replace\bbl@tempa{enumerate.}{}%
3081      \def\bbl@toreplace{#2}%
3082      \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
3083      \bbl@replace\bbl@toreplace{[}{\csname the}%
3084      \bbl@replace\bbl@toreplace{]}{\endcsname{}}%
3085      \toks@\expandafter{\bbl@toreplace}%
3086      % TODO. Execute only once:
3087      \bbl@exp{%
3088        \\\bbl@add\<extras\languagename>{%
3089          \\\babel@save\<labelenum\romannumeral\bbl@tempa>%
3090          \def\<labelenum\romannumeral\bbl@tempa>{\the\toks@}}%
3091        \\\bbl@toglobal\<extras\languagename>}%
3092    \fi
3093  \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.

```
3094 \def\bbl@chaptype{chapter}
3095 \ifx\@makechapterhead\@undefined
3096   \let\bbl@patchchapter\relax
3097 \else\ifx\thechapter\@undefined
3098   \let\bbl@patchchapter\relax
3099 \else\ifx\ps@headings\@undefined
3100   \let\bbl@patchchapter\relax
3101 \else
3102   \def\bbl@patchchapter{%
3103     \global\let\bbl@patchchapter\relax
3104     \gdef\bbl@chfmt{%
3105       \bbl@ifunset{bbl@\bbl@chaptype fmt@\languagename}%
3106         {\@chapapp\space\thechapter}
3107         {\@nameuse{bbl@\bbl@chaptype fmt@\languagename}}}%
3108     \bbl@add\appendix{\def\bbl@chaptype{appendix}}% Not harmful, I hope
3109     \bbl@sreplace\ps@headings{\@chapapp\ \thechapter}{\bbl@chfmt}%
3110     \bbl@sreplace\chaptermark{\@chapapp\ \thechapter}{\bbl@chfmt}%
3111     \bbl@sreplace\@makechapterhead{\@chapapp\space\thechapter}{\bbl@chfmt}%
3112     \bbl@toglobal\appendix
3113     \bbl@toglobal\ps@headings
3114     \bbl@toglobal\chaptermark
3115     \bbl@toglobal\@makechapterhead}
3116   \let\bbl@patchappendix\bbl@patchchapter
3117 \fi\fi\fi
3118 \ifx\@part\@undefined
3119   \let\bbl@patchpart\relax
3120 \else
3121   \def\bbl@patchpart{%
3122     \global\let\bbl@patchpart\relax
3123     \gdef\bbl@partformat{%
3124       \bbl@ifunset{bbl@partfmt@\languagename}%
3125         {\partname\nobreakspace\thepart}
3126         {\@nameuse{bbl@partfmt@\languagename}}}%
3127     \bbl@sreplace\@part{\partname\nobreakspace\thepart}{\bbl@partformat}%
```

```
3128       \bbl@toglobal\@part}
3129 \fi
```

Date. TODO. Document

```
3130 % Arguments are _not_ protected.
3131 \let\bbl@calendar\@empty
3132 \DeclareRobustCommand\localedate[1][]{\bbl@localedate{#1}}
3133 \def\bbl@localedate#1#2#3#4{%
3134    \begingroup
3135      \ifx\@empty#1\@empty\else
3136        \let\bbl@ld@calendar\@empty
3137        \let\bbl@ld@variant\@empty
3138        \edef\bbl@tempa{\zap@space#1 \@empty}%
3139        \def\bbl@tempb##1=##2\@@{\@namedef{bbl@ld@##1}{##2}}%
3140        \bbl@foreach\bbl@tempa{\bbl@tempb##1\@@}%
3141        \edef\bbl@calendar{%
3142          \bbl@ld@calendar
3143          \ifx\bbl@ld@variant\@empty\else
3144            .\bbl@ld@variant
3145          \fi}%
3146        \bbl@replace\bbl@calendar{gregorian}{}%
3147      \fi
3148      \bbl@cased
3149        {\@nameuse{bbl@date@\languagename @\bbl@calendar}{#2}{#3}{#4}}%
3150    \endgroup}
3151 % eg: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
3152 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
3153    \bbl@trim@def\bbl@tempa{#1.#2}%
3154    \bbl@ifsamestring{\bbl@tempa}{months.wide}%        to savedate
3155      {\bbl@trim@def\bbl@tempa{#3}%
3156       \bbl@trim\toks@{#5}%
3157       \@temptokena\expandafter{\bbl@savedate}%
3158       \bbl@exp{%    Reverse order - in ini last wins
3159         \def\\\bbl@savedate{%
3160           \\\SetString\<month\romannumeral\bbl@tempa#6name>{\the\toks@}%
3161           \the\@temptokena}}}%
3162    {\bbl@ifsamestring{\bbl@tempa}{date.long}%        defined now
3163      {\lowercase{\def\bbl@tempb{#6}}%
3164       \bbl@trim@def\bbl@toreplace{#5}%
3165       \bbl@TG@@date
3166       \bbl@ifunset{bbl@date@\languagename @}%
3167         {\bbl@exp{% TODO. Move to a better place.
3168            \gdef\<\languagename date>{\\\protect\<\languagename date >}%
3169            \gdef\<\languagename date >####1####2####3{%
3170              \\\bbl@usedategrouptrue
3171              \<bbl@ensure@\languagename>{%
3172                \\\localedate{####1}{####2}{####3}}}%
3173            \\\bbl@add\\\bbl@savetoday{%
3174              \\\SetString\\\today{%
3175                \<\languagename date>%
3176                  {\\\the\year}{\\\the\month}{\\\the\day}}}}}%
3177         {}%
3178       \global\bbl@csarg\let{date@\languagename @}\bbl@toreplace
3179       \ifx\bbl@tempb\@empty\else
3180         \global\bbl@csarg\let{date@\languagename @\bbl@tempb}\bbl@toreplace
3181       \fi}%
3182      {}}}
```

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"

134

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).

```
3183 \let\bbl@calendar\@empty
3184 \newcommand\BabelDateSpace{\nobreakspace}
3185 \newcommand\BabelDateDot{.\@}  % TODO. \let instead of repeating
3186 \newcommand\BabelDated[1]{{\number#1}}
3187 \newcommand\BabelDatedd[1]{{\ifnum#1<10 0\fi\number#1}}
3188 \newcommand\BabelDateM[1]{{\number#1}}
3189 \newcommand\BabelDateMM[1]{{\ifnum#1<10 0\fi\number#1}}
3190 \newcommand\BabelDateMMMM[1]{{%
3191   \csname month\romannumeral#1\bbl@calendar name\endcsname}}%
3192 \newcommand\BabelDatey[1]{{\number#1}}%
3193 \newcommand\BabelDateyy[1]{{%
3194   \ifnum#1<10 0\number#1 %
3195   \else\ifnum#1<100 \number#1 %
3196   \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3197   \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
3198   \else
3199     \bbl@error
3200       {Currently two-digit years are restricted to the\\
3201        range 0-9999.}%
3202       {There is little you can do. Sorry.}%
3203   \fi\fi\fi\fi}}
3204 \newcommand\BabelDateyyyy[1]{{\number#1}} % TODO - add leading 0
3205 \def\bbl@replace@finish@iii#1{%
3206   \bbl@exp{\def\\#1####1####2####3{\the\toks@}}}
3207 \def\bbl@TG@@date{%
3208   \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace{}}%
3209   \bbl@replace\bbl@toreplace{[.]}{\BabelDateDot{}}%
3210   \bbl@replace\bbl@toreplace{[d]}{\BabelDated{####3}}%
3211   \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{####3}}%
3212   \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{####2}}%
3213   \bbl@replace\bbl@toreplace{[MM]}{\BabelDateMM{####2}}%
3214   \bbl@replace\bbl@toreplace{[MMMM]}{\BabelDateMMMM{####2}}%
3215   \bbl@replace\bbl@toreplace{[y]}{\BabelDatey{####1}}%
3216   \bbl@replace\bbl@toreplace{[yy]}{\BabelDateyy{####1}}%
3217   \bbl@replace\bbl@toreplace{[yyyy]}{\BabelDateyyyy{####1}}%
3218   \bbl@replace\bbl@toreplace{[y|}{\bbl@datecntr[####1|}%
3219   \bbl@replace\bbl@toreplace{[m|}{\bbl@datecntr[####2|}%
3220   \bbl@replace\bbl@toreplace{[d|}{\bbl@datecntr[####3|}%
3221   \bbl@replace@finish@iii\bbl@toreplace}
3222 \def\bbl@datecntr{\expandafter\bbl@xdatecntr\expandafter}
3223 \def\bbl@xdatecntr[#1|#2]{\localenumeral{#2}{#1}}
```

Transforms.

```
3224 \let\bbl@release@transforms\@empty
3225 \@namedef{bbl@inikv@transforms.prehyphenation}{%
3226   \bbl@transforms\babelprehyphenation}
3227 \@namedef{bbl@inikv@transforms.posthyphenation}{%
3228   \bbl@transforms\babelposthyphenation}
3229 \def\bbl@transforms@aux#1#2#3,#4\relax{#1{#2}{#3}{#4}}
3230 \begingroup %  A hack. TODO. Don't require an specific order
3231   \catcode`\%=12
3232   \catcode`\&=14
3233   \gdef\bbl@transforms#1#2#3{&%
3234     \ifx\bbl@KVP@transforms\@nil\else
3235       \directlua{
3236         str = [==[#2]==]
```

135

```
3237        str = str:gsub('%.%d+%.%d+$', '')
3238        tex.print([[\def\string\babeltempa{]] .. str .. [[}]])
3239      }&%
3240      \bbl@xin@{,\babeltempa,}{,\bbl@KVP@transforms,}&%
3241      \ifin@
3242        \in@{.0$}{#2$}&%
3243        \ifin@
3244          \g@addto@macro\bbl@release@transforms{&%
3245            \relax\bbl@transforms@aux#1{\languagename}{#3}}&%
3246        \else
3247          \g@addto@macro\bbl@release@transforms{, {#3}}&%
3248        \fi
3249      \fi
3250    \fi}
3251 \endgroup
```

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

```
3252 \def\bbl@provide@lsys#1{%
3253    \bbl@ifunset{bbl@lname@#1}%
3254      {\bbl@load@info{#1}}%
3255      {}%
3256    \bbl@csarg\let{lsys@#1}\@empty
3257    \bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}{}%
3258    \bbl@ifunset{bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{}%
3259    \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
3260    \bbl@ifunset{bbl@lname@#1}{}%
3261      {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
3262    \ifcase\bbl@engine\or\or
3263      \bbl@ifunset{bbl@prehc@#1}{}%
3264        {\bbl@exp{\\\bbl@ifblank{\bbl@cs{prehc@#1}}}%
3265          {}%
3266          {\ifx\bbl@xenohyph\@undefined
3267            \let\bbl@xenohyph\bbl@xenohyph@d
3268            \ifx\AtBeginDocument\@notprerr
3269              \expandafter\@secondoftwo  % to execute right now
3270            \fi
3271            \AtBeginDocument{%
3272              \bbl@patchfont{\bbl@xenohyph}%
3273              \expandafter\selectlanguage\expandafter{\languagename}}%
3274          \fi}}%
3275    \fi
3276    \bbl@csarg\bbl@toglobal{lsys@#1}}
3277 \def\bbl@xenohyph@d{%
3278    \bbl@ifset{bbl@prehc@\languagename}%
3279      {\ifnum\hyphenchar\font=\defaulthyphenchar
3280        \iffontchar\font\bbl@cl{prehc}\relax
3281          \hyphenchar\font\bbl@cl{prehc}\relax
3282        \else\iffontchar\font"200B
3283          \hyphenchar\font"200B
3284        \else
3285          \bbl@warning
3286            {Neither 0 nor ZERO WIDTH SPACE are available\\%
3287             in the current font, and therefore the hyphen\\%
3288             will be printed. Try changing the fontspec's\\%
3289             'HyphenChar' to another value, but be aware\\%
3290             this setting is not safe (see the manual)}%
3291          \hyphenchar\font\defaulthyphenchar
3292        \fi\fi
```

```
3293      \fi}%
3294    {\hyphenchar\font\defaulthyphenchar}}
3295  % \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).

```
3296 \def\bbl@load@info#1{%
3297   \def\BabelBeforeIni##1##2{%
3298     \begingroup
3299       \bbl@read@ini{##1}0%
3300       \endinput          % babel- .tex may contain onlypreamble's
3301     \endgroup}%              boxed, to avoid extra spaces:
3302   {\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.

```
3303 \def\bbl@setdigits#1#2#3#4#5{%
3304   \bbl@exp{%
3305     \def\<\languagename digits>####1{%         ie, \langdigits
3306       \<bbl@digits@\languagename>####1\\\@nil}%
3307     \let\<bbl@cntr@digits@\languagename>\<\languagename digits>%
3308     \def\<\languagename counter>####1{%        ie, \langcounter
3309       \\\expandafter\<bbl@counter@\languagename>%
3310       \\\csname c@####1\endcsname}%
3311     \def\<bbl@counter@\languagename>####1{% ie, \bbl@counter@lang
3312       \\\expandafter\<bbl@digits@\languagename>%
3313       \\\number####1\\\@nil}}%
3314   \def\bbl@tempa##1##2##3##4##5{%
3315     \bbl@exp{%    Wow, quite a lot of hashes! :-(
3316       \def\<bbl@digits@\languagename>########1{%
3317         \\\ifx########1\\\@nil              % ie, \bbl@digits@lang
3318         \\\else
3319           \\\ifx0########1#1%
3320           \\\else\\\ifx1########1#2%
3321           \\\else\\\ifx2########1#3%
3322           \\\else\\\ifx3########1#4%
3323           \\\else\\\ifx4########1#5%
3324           \\\else\\\ifx5########1##1%
3325           \\\else\\\ifx6########1##2%
3326           \\\else\\\ifx7########1##3%
3327           \\\else\\\ifx8########1##4%
3328           \\\else\\\ifx9########1##5%
3329           \\\else########1%
3330           \\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi
3331           \\\expandafter\<bbl@digits@\languagename>%
3332         \\\fi}}}%
3333   \bbl@tempa}
```

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

```
3334 \def\bbl@buildifcase#1 {% Returns \bbl@tempa, requires \toks@={}
3335   \ifx\\#1%               % \\ before, in case #1 is multiletter
3336     \bbl@exp{%
3337       \def\\\bbl@tempa####1{%
3338         \<ifcase>####1\space\the\toks@\<else>\\\@ctrerr\<fi>}}%
3339   \else
3340     \toks@\expandafter{\the\toks@\or #1}%
```

```
3341      \expandafter\bbl@buildifcase
3342   \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).

```
3343 \newcommand\localenumeral[2]{\bbl@cs{cntr@#1@\languagename}{#2}}
3344 \def\bbl@localecntr#1#2{\localenumeral{#2}{#1}}
3345 \newcommand\localecounter[2]{%
3346   \expandafter\bbl@localecntr
3347   \expandafter{\number\csname c@#2\endcsname}{#1}}
3348 \def\bbl@alphnumeral#1#2{%
3349   \expandafter\bbl@alphnumeral@i\number#2 76543210\@@{#1}}
3350 \def\bbl@alphnumeral@i#1#2#3#4#5#6#7#8\@@#9{%
3351   \ifcase\@car#8\@nil\or    % Currenty <10000, but prepared for bigger
3352     \bbl@alphnumeral@ii{#9}000000#1\or
3353     \bbl@alphnumeral@ii{#9}00000#1#2\or
3354     \bbl@alphnumeral@ii{#9}0000#1#2#3\or
3355     \bbl@alphnumeral@ii{#9}000#1#2#3#4\else
3356     \bbl@alphnum@invalid{>9999}%
3357   \fi}
3358 \def\bbl@alphnumeral@ii#1#2#3#4#5#6#7#8{%
3359   \bbl@ifunset{bbl@cntr@#1.F.\number#5#6#7#8@\languagename}%
3360     {\bbl@cs{cntr@#1.4@\languagename}#5%
3361      \bbl@cs{cntr@#1.3@\languagename}#6%
3362      \bbl@cs{cntr@#1.2@\languagename}#7%
3363      \bbl@cs{cntr@#1.1@\languagename}#8%
3364      \ifnum#6#7#8>\z@ % TODO. An ad hoc rule for Greek. Ugly.
3365        \bbl@ifunset{bbl@cntr@#1.S.321@\languagename}{}%
3366          {\bbl@cs{cntr@#1.S.321@\languagename}}%
3367      \fi}%
3368     {\bbl@cs{cntr@#1.F.\number#5#6#7#8@\languagename}}}
3369 \def\bbl@alphnum@invalid#1{%
3370   \bbl@error{Alphabetic numeral too large (#1)}%
3371     {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.

```
3372 \newcommand\localeinfo[1]{%
3373   \bbl@ifunset{bbl@\csname bbl@info@#1\endcsname @\languagename}%
3374     {\bbl@error{I've found no info for the current locale.\\%
3375                 The corresponding ini file has not been loaded\\%
3376                 Perhaps it doesn't exist}%
3377                {See the manual for details.}}%
3378     {\bbl@cs{\csname bbl@info@#1\endcsname @\languagename}}}
3379 % \@namedef{bbl@info@name.locale}{lcname}
3380 \@namedef{bbl@info@tag.ini}{lini}
3381 \@namedef{bbl@info@name.english}{elname}
3382 \@namedef{bbl@info@name.opentype}{lname}
3383 \@namedef{bbl@info@tag.bcp47}{tbcp}
3384 \@namedef{bbl@info@language.tag.bcp47}{lbcp}
3385 \@namedef{bbl@info@tag.opentype}{lotf}
3386 \@namedef{bbl@info@script.name}{esname}
3387 \@namedef{bbl@info@script.name.opentype}{sname}
3388 \@namedef{bbl@info@script.tag.bcp47}{sbcp}
3389 \@namedef{bbl@info@script.tag.opentype}{sotf}
3390 \let\bbl@ensureinfo\@gobble
```

```
3391 \newcommand\BabelEnsureInfo{%
3392   \ifx\InputIfFileExists\@undefined\else
3393     \def\bbl@ensureinfo##1{%
3394       \bbl@ifunset{bbl@lname@##1}{\bbl@load@info{##1}}{}}%
3395   \fi
3396   \bbl@foreach\bbl@loaded{{%
3397     \def\languagename{##1}%
3398     \bbl@ensureinfo{##1}}}}
```

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.

```
3399 \newcommand\getlocaleproperty{%
3400   \@ifstar\bbl@getproperty@s\bbl@getproperty@x}
3401 \def\bbl@getproperty@s#1#2#3{%
3402   \let#1\relax
3403   \def\bbl@elt##1##2##3{%
3404     \bbl@ifsamestring{##1/##2}{#3}%
3405       {\providecommand#1{##3}%
3406        \def\bbl@elt####1####2####3{}}%
3407       {}}%
3408   \bbl@cs{inidata@#2}}%
3409 \def\bbl@getproperty@x#1#2#3{%
3410   \bbl@getproperty@s{#1}{#2}{#3}%
3411   \ifx#1\relax
3412     \bbl@error
3413       {Unknown key for locale '#2':\\%
3414        #3\\%
3415        \string#1 will be set to \relax}%
3416       {Perhaps you misspelled it.}%
3417   \fi}
3418 \let\bbl@ini@loaded\@empty
3419 \newcommand\LocaleForEach{\bbl@foreach\bbl@ini@loaded}
```

9   Adjusting the Babel bahavior

A generic high level inteface is provided to adjust some global and general settings.

```
3420 \newcommand\babeladjust[1]{%  TODO. Error handling.
3421   \bbl@forkv{#1}{%
3422     \bbl@ifunset{bbl@ADJ@##1@##2}%
3423       {\bbl@cs{ADJ@##1}{##2}}%
3424       {\bbl@cs{ADJ@##1@##2}}}}
3425 %
3426 \def\bbl@adjust@lua#1#2{%
3427   \ifvmode
3428     \ifnum\currentgrouplevel=\z@
3429       \directlua{ Babel.#2 }%
3430       \expandafter\expandafter\expandafter\@gobble
3431     \fi
3432   \fi
3433   {\bbl@error   % The error is gobbled if everything went ok.
3434     {Currently, #1 related features can be adjusted only\\%
3435      in the main vertical list.}%
3436     {Maybe things change in the future, but this is what it is.}}}
3437 \@namedef{bbl@ADJ@bidi.mirroring@on}{%
3438   \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
3439 \@namedef{bbl@ADJ@bidi.mirroring@off}{%
3440   \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
```

```
3441 \@namedef{bbl@ADJ@bidi.text@on}{%
3442   \bbl@adjust@lua{bidi}{bidi_enabled=true}}
3443 \@namedef{bbl@ADJ@bidi.text@off}{%
3444   \bbl@adjust@lua{bidi}{bidi_enabled=false}}
3445 \@namedef{bbl@ADJ@bidi.mapdigits@on}{%
3446   \bbl@adjust@lua{bidi}{digits_mapped=true}}
3447 \@namedef{bbl@ADJ@bidi.mapdigits@off}{%
3448   \bbl@adjust@lua{bidi}{digits_mapped=false}}
3449 %
3450 \@namedef{bbl@ADJ@linebreak.sea@on}{%
3451   \bbl@adjust@lua{linebreak}{sea_enabled=true}}
3452 \@namedef{bbl@ADJ@linebreak.sea@off}{%
3453   \bbl@adjust@lua{linebreak}{sea_enabled=false}}
3454 \@namedef{bbl@ADJ@linebreak.cjk@on}{%
3455   \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
3456 \@namedef{bbl@ADJ@linebreak.cjk@off}{%
3457   \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
3458 \@namedef{bbl@ADJ@justify.arabic@on}{%
3459   \bbl@adjust@lua{linebreak}{arabic.justify_enabled=true}}
3460 \@namedef{bbl@ADJ@justify.arabic@off}{%
3461   \bbl@adjust@lua{linebreak}{arabic.justify_enabled=false}}
3462 %
3463 \def\bbl@adjust@layout#1{%
3464   \ifvmode
3465     #1%
3466     \expandafter\@gobble
3467   \fi
3468   {\bbl@error    % The error is gobbled if everything went ok.
3469     {Currently, layout related features can be adjusted only\\%
3470      in vertical mode.}%
3471    {Maybe things change in the future, but this is what it is.}}}
3472 \@namedef{bbl@ADJ@layout.tabular@on}{%
3473   \bbl@adjust@layout{\let\@tabular\bbl@NL@@tabular}}
3474 \@namedef{bbl@ADJ@layout.tabular@off}{%
3475   \bbl@adjust@layout{\let\@tabular\bbl@OL@@tabular}}
3476 \@namedef{bbl@ADJ@layout.lists@on}{%
3477   \bbl@adjust@layout{\let\list\bbl@NL@list}}
3478 \@namedef{bbl@ADJ@layout.lists@off}{%
3479   \bbl@adjust@layout{\let\list\bbl@OL@list}}
3480 \@namedef{bbl@ADJ@hyphenation.extra@on}{%
3481   \bbl@activateposthyphen}
3482 %
3483 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
3484   \bbl@bcpallowedtrue}
3485 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
3486   \bbl@bcpallowedfalse}
3487 \@namedef{bbl@ADJ@autoload.bcp47.prefix}#1{%
3488   \def\bbl@bcp@prefix{#1}}
3489 \def\bbl@bcp@prefix{bcp47-}
3490 \@namedef{bbl@ADJ@autoload.options}#1{%
3491   \def\bbl@autoload@options{#1}}
3492 \let\bbl@autoload@bcpoptions\@empty
3493 \@namedef{bbl@ADJ@autoload.bcp47.options}#1{%
3494   \def\bbl@autoload@bcpoptions{#1}}
3495 \newif\ifbbl@bcptoname
3496 \@namedef{bbl@ADJ@bcp47.toname@on}{%
3497   \bbl@bcptonametrue
3498   \BabelEnsureInfo}
3499 \@namedef{bbl@ADJ@bcp47.toname@off}{%
```

```
3500    \bbl@bcptonamefalse}
3501 \@namedef{bbl@ADJ@prehyphenation.disable@nohyphenation}{%
3502    \directlua{ Babel.ignore_pre_char = function(node)
3503       return (node.lang == \the\csname l@nohyphenation\endcsname)
3504     end }}
3505 \@namedef{bbl@ADJ@prehyphenation.disable@off}{%
3506    \directlua{ Babel.ignore_pre_char = function(node)
3507       return false
3508     end }}
3509 \@namedef{bbl@ADJ@select.write@shift}{%
3510    \let\bbl@restorelastskip\relax
3511    \def\bbl@savelastskip{%
3512     \let\bbl@restorelastskip\relax
3513     \ifvmode
3514       \ifdim\lastskip=\z@
3515         \let\bbl@restorelastskip\nobreak
3516       \else
3517         \bbl@exp{%
3518           \def\\\bbl@restorelastskip{%
3519             \skip@=\the\lastskip
3520             \\\nobreak \vskip-\skip@ \vskip\skip@}}%
3521       \fi
3522     \fi}}
3523 \@namedef{bbl@ADJ@select.write@keep}{%
3524    \let\bbl@restorelastskip\relax
3525    \let\bbl@savelastskip\relax}
3526 \@namedef{bbl@ADJ@select.write@omit}{%
3527    \let\bbl@restorelastskip\relax
3528    \def\bbl@savelastskip##1\bbl@restorelastskip{}}
```

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

```
3529 \ifx\directlua\@undefined\else
3530    \ifx\bbl@luapatterns\@undefined
3531      \input luababel.def
3532    \fi
3533 \fi
```

Continue with LaTeX.

```
3534 ⟨/package | core⟩
3535 ⟨*package⟩
```

9.1 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.

```
3536 ⟨⟨*More package options⟩⟩ ≡
3537 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
3538 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
3539 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
3540 ⟨⟨/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.

```
3541 \bbl@trace{Cross referencing macros}
3542 \ifx\bbl@opt@safe\@empty\else
3543   \def\@newl@bel#1#2#3{%
3544     {\@safe@activestrue
3545      \bbl@ifunset{#1@#2}%
3546         \relax
3547         {\gdef\@multiplelabels{%
3548            \@latex@warning@no@line{There were multiply-defined labels}}%
3549          \@latex@warning@no@line{Label `#2' multiply defined}}%
3550      \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.

```
3551 \CheckCommand*\@testdef[3]{%
3552   \def\reserved@a{#3}%
3553   \expandafter\ifx\csname#1@#2\endcsname\reserved@a
3554   \else
3555     \@tempswatrue
3556   \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.

```
3557   \def\@testdef#1#2#3{%  TODO. With @samestring?
3558     \@safe@activestrue
3559     \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
3560     \def\bbl@tempb{#3}%
3561     \@safe@activesfalse
3562     \ifx\bbl@tempa\relax
3563     \else
3564       \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
3565     \fi
3566     \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
3567     \ifx\bbl@tempa\bbl@tempb
3568     \else
3569       \@tempswatrue
3570     \fi}
3571 \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.

```
3572 \bbl@xin@{R}\bbl@opt@safe
3573 \ifin@
3574   \bbl@redefinerobust\ref#1{%
3575     \@safe@activestrue\org@ref{#1}\@safe@activesfalse}
3576   \bbl@redefinerobust\pageref#1{%
3577     \@safe@activestrue\org@pageref{#1}\@safe@activesfalse}
3578 \else
3579   \let\org@ref\ref
3580   \let\org@pageref\pageref
3581 \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.

```
3582 \bbl@xin@{B}\bbl@opt@safe
3583 \ifin@
3584   \bbl@redefine\@citex[#1]#2{%
3585     \@safe@activestrue\edef\@tempa{#2}\@safe@activesfalse
3586     \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.

```
3587   \AtBeginDocument{%
3588     \@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.)

```
3589     \def\@citex[#1][#2]#3{%
3590       \@safe@activestrue\edef\@tempa{#3}\@safe@activesfalse
3591       \org@@citex[#1][#2]{\@tempa}}%
3592     }{}}
```

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

```
3593   \AtBeginDocument{%
3594     \@ifpackageloaded{cite}{%
3595       \def\@citex[#1]#2{%
3596         \@safe@activestrue\org@@citex[#1]{#2}\@safe@activesfalse}%
3597       }{}}
```

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

```
3598 \bbl@redefine\nocite#1{%
3599   \@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.

```
3600 \bbl@redefine\bibcite{%
3601   \bbl@cite@choice
3602   \bibcite}
```

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

```
3603 \def\bbl@bibcite#1#2{%
3604   \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.

```
3605 \def\bbl@cite@choice{%
3606   \global\let\bibcite\bbl@bibcite
```

143

```
3607    \@ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{}%
3608    \@ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{}%
3609    \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.

```
3610    \AtBeginDocument{\bbl@cite@choice}
```

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

```
3611    \bbl@redefine\@bibitem#1{%
3612      \@safe@activestrue\org@@bibitem{#1}\@safe@activesfalse}
3613 \else
3614    \let\org@nocite\nocite
3615    \let\org@@citex\@citex
3616    \let\org@bibcite\bibcite
3617    \let\org@@bibitem\@bibitem
3618 \fi
```

9.2   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.

```
3619 \bbl@trace{Marks}
3620 \IfBabelLayout{sectioning}
3621    {\ifx\bbl@opt@headfoot\@nnil
3622      \g@addto@macro\@resetactivechars{%
3623        \set@typeset@protect
3624        \expandafter\select@language@x\expandafter{\bbl@main@language}%
3625        \let\protect\noexpand
3626        \ifcase\bbl@bidimode\else % Only with bidi. See also above
3627          \edef\thepage{%
3628            \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
3629        \fi}%
3630      \fi}
3631    {\ifbbl@single\else
3632      \bbl@ifunset{markright }\bbl@redefine\bbl@redefinerobust
3633      \markright#1{%
3634        \bbl@ifblank{#1}%
3635          {\org@markright{}}%
3636          {\toks@{#1}%
3637            \bbl@exp{%
3638              \\\org@markright{\\\protect\\\foreignlanguage{\languagename}%
3639                {\\\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.)

```
3640      \ifx\@mkboth\markboth
3641        \def\bbl@tempc{\let\@mkboth\markboth}
3642      \else
3643        \def\bbl@tempc{}
```

```
3644        \fi
3645      \bbl@ifunset{markboth }\bbl@redefine\bbl@redefinerobust
3646      \markboth#1#2{%
3647        \protected@edef\bbl@tempb##1{%
3648          \protect\foreignlanguage
3649          {\languagename}{\protect\bbl@restore@actives##1}}%
3650        \bbl@ifblank{#1}%
3651          {\toks@{}}%
3652          {\toks@\expandafter{\bbl@tempb{#1}}}%
3653        \bbl@ifblank{#2}%
3654          {\@temptokena{}}%
3655          {\@temptokena\expandafter{\bbl@tempb{#2}}}%
3656        \bbl@exp{\\\org@markboth{\the\toks@}{\the\@temptokena}}}
3657        \bbl@tempc
3658      \fi}  % end ifbbl@single, end \IfBabelLayout
```

9.3 Preventing clashes with other packages

9.3.1 `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.

```
3659 \bbl@trace{Preventing clashes with other packages}
3660 \bbl@xin@{R}\bbl@opt@safe
3661 \ifin@
3662   \AtBeginDocument{%
3663     \@ifpackageloaded{ifthen}{%
3664       \bbl@redefine@long\ifthenelse#1#2#3{%
3665         \let\bbl@temp@pref\pageref
3666         \let\pageref\org@pageref
3667         \let\bbl@temp@ref\ref
3668         \let\ref\org@ref
3669         \@safe@activestrue
3670         \org@ifthenelse{#1}%
3671           {\let\pageref\bbl@temp@pref
3672            \let\ref\bbl@temp@ref
3673            \@safe@activesfalse
3674            #2}%
3675           {\let\pageref\bbl@temp@pref
3676            \let\ref\bbl@temp@ref
3677            \@safe@activesfalse
3678            #3}%
3679       }%
3680     }{}%
3681   }
```

9.3.2 varioref

\@@vpageref
\vrefpagenum
\Ref

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

```
3682 \AtBeginDocument{%
3683   \@ifpackageloaded{varioref}{%
3684     \bbl@redefine\@@vpageref#1[#2]#3{%
3685       \@safe@activestrue
3686       \org@@@vpageref{#1}[#2]{#3}%
3687       \@safe@activesfalse}%
3688     \bbl@redefine\vrefpagenum#1#2{%
3689       \@safe@activestrue
3690       \org@vrefpagenum{#1}{#2}%
3691       \@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.

```
3692     \expandafter\def\csname Ref \endcsname#1{%
3693       \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
3694   }{}%
3695   }
3696 \fi
```

9.3.3 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.

```
3697 \AtEndOfPackage{%
3698   \AtBeginDocument{%
3699     \@ifpackageloaded{hhline}%
3700       {\expandafter\ifx\csname normal@char\string:\endcsname\relax
3701        \else
3702          \makeatletter
3703          \def\@currname{hhline}\input{hhline.sty}\makeatother
3704        \fi}%
3705       {}}}
```

\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.

```
3706 \def\substitutefontfamily#1#2#3{%
3707   \lowercase{\immediate\openout15=#1#2.fd\relax}%
3708   \immediate\write15{%
3709     \string\ProvidesFile{#1#2.fd}%
3710     [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
3711      \space generated font description file]^^J
3712     \string\DeclareFontFamily{#1}{#2}{}^^J
3713     \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{}^^J
3714     \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{}^^J
3715     \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{}^^J
3716     \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{}^^J
```

```
3717    \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{}^^J
3718    \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{}^^J
3719    \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{}^^J
3720    \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{}^^J
3721    }%
3722  \closeout15
3723  }
3724 \@onlypreamble\substitutefontfamily
```

9.4   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

```
3725 \bbl@trace{Encoding and fonts}
3726 \newcommand\BabelNonASCII{LGR,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU}
3727 \newcommand\BabelNonText{TS1,T3,TS3}
3728 \let\org@TeX\TeX
3729 \let\org@LaTeX\LaTeX
3730 \let\ensureascii\@firstofone
3731 \AtBeginDocument{%
3732   \def\@elt#1{,#1,}%
3733   \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3734   \let\@elt\relax
3735   \let\bbl@tempb\@empty
3736   \def\bbl@tempc{OT1}%
3737   \bbl@foreach\BabelNonASCII{% LGR loaded in a non-standard way
3738     \bbl@ifunset{T@#1}{}{\def\bbl@tempb{#1}}}%
3739   \bbl@foreach\bbl@tempa{%
3740     \bbl@xin@{#1}{\BabelNonASCII}%
3741     \ifin@
3742       \def\bbl@tempb{#1}% Store last non-ascii
3743     \else\bbl@xin@{#1}{\BabelNonText}% Pass
3744       \ifin@\else
3745         \def\bbl@tempc{#1}% Store last ascii
3746       \fi
3747     \fi}%
3748   \ifx\bbl@tempb\@empty\else
3749     \bbl@xin@{,\cf@encoding,}{,\BabelNonASCII,\BabelNonText,}%
3750     \ifin@\else
3751       \edef\bbl@tempc{\cf@encoding}% The default if ascii wins
3752     \fi
3753     \edef\ensureascii#1{%
3754       {\noexpand\fontencoding{\bbl@tempc}\noexpand\selectfont#1}}%
3755     \DeclareTextCommandDefault{\TeX}{\ensureascii{\org@TeX}}%
3756     \DeclareTextCommandDefault{\LaTeX}{\ensureascii{\org@LaTeX}}%
3757   \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.

```
3758 \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.

```
3759 \AtBeginDocument{%
3760   \@ifpackageloaded{fontspec}%
3761     {\xdef\latinencoding{%
3762       \ifx\UTFencname\@undefined
3763         EU\ifcase\bbl@engine\or2\or1\fi
3764       \else
3765         \UTFencname
3766       \fi}}%
3767     {\gdef\latinencoding{OT1}%
3768      \ifx\cf@encoding\bbl@t@one
3769        \xdef\latinencoding{\bbl@t@one}%
3770      \else
3771        \def\@elt#1{,#1,}%
3772        \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3773        \let\@elt\relax
3774        \bbl@xin@{,T1,}\bbl@tempa
3775        \ifin@
3776          \xdef\latinencoding{\bbl@t@one}%
3777        \fi
3778      \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.

```
3779 \DeclareRobustCommand{\latintext}{%
3780   \fontencoding{\latinencoding}\selectfont
3781   \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.

```
3782 \ifx\@undefined\DeclareTextFontCommand
3783   \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
3784 \else
3785   \DeclareTextFontCommand{\textlatin}{\latintext}
3786 \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).

```
3787 \bbl@ifformatlater{2021-06-01}%
3788   {\def\bbl@patchfont#1{\AddToHook{selectfont}{#1}}}
3789   {\def\bbl@patchfont#1{%
3790     \expandafter\bbl@add\csname selectfont \endcsname{#1}%
3791     \expandafter\bbl@toglobal\csname selectfont \endcsname}}
```

9.5   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.

```
3792 \bbl@trace{Loading basic (internal) bidi support}
3793 \ifodd\bbl@engine
3794 \else % TODO. Move to txtbabel
3795   \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
3796     \bbl@error
3797       {The bidi method 'basic' is available only in\\%
3798        luatex. I'll continue with 'bidi=default', so\\%
3799        expect wrong results}%
3800       {See the manual for further details.}%
3801     \let\bbl@beforeforeign\leavevmode
3802     \AtEndOfPackage{%
3803       \EnableBabelHook{babel-bidi}%
3804       \bbl@xebidipar}
3805   \fi\fi
3806 \def\bbl@loadxebidi#1{%
3807   \ifx\RTLfootnotetext\@undefined
3808     \AtEndOfPackage{%
3809       \EnableBabelHook{babel-bidi}%
3810       \ifx\fontspec\@undefined
3811         \bbl@loadfontspec % bidi needs fontspec
3812       \fi
3813       \usepackage#1{bidi}}%
3814   \fi}
3815   \ifnum\bbl@bidimode>200
3816     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
3817       \bbl@tentative{bidi=bidi}
3818       \bbl@loadxebidi{}
3819     \or
3820       \bbl@loadxebidi{[rldocument]}
3821     \or
3822       \bbl@loadxebidi{}
3823     \fi
3824   \fi
3825 \fi
3826 % TODO? Separate:
3827 \ifnum\bbl@bidimode=\@ne
3828   \let\bbl@beforeforeign\leavevmode
3829   \ifodd\bbl@engine
3830     \newattribute\bbl@attr@dir
3831     \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
3832     \bbl@exp{\output{\bodydir\pagedir\the\output}}
3833   \fi
3834   \AtEndOfPackage{%
3835     \EnableBabelHook{babel-bidi}%
3836     \ifodd\bbl@engine\else
```

```
3837        \bbl@xebidipar
3838      \fi}
3839 \fi
```

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

```
3840 \bbl@trace{Macros to switch the text direction}
3841 \def\bbl@alscripts{,Arabic,Syriac,Thaana,}
3842 \def\bbl@rscripts{% TODO. Base on codes ??
3843    ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
3844    Old Hungarian,Old Hungarian,Lydian,Mandaean,Manichaean,%
3845    Manichaean,Meroitic Cursive,Meroitic,Old North Arabian,%
3846    Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
3847    Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
3848    Old South Arabian,}%
3849 \def\bbl@provide@dirs#1{%
3850    \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
3851    \ifin@
3852      \global\bbl@csarg\chardef{wdir@#1}\@ne
3853      \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
3854      \ifin@
3855        \global\bbl@csarg\chardef{wdir@#1}\tw@  % useless in xetex
3856      \fi
3857    \else
3858      \global\bbl@csarg\chardef{wdir@#1}\z@
3859    \fi
3860    \ifodd\bbl@engine
3861      \bbl@csarg\ifcase{wdir@#1}%
3862        \directlua{ Babel.locale_props[\the\localeid].textdir = 'l' }%
3863      \or
3864        \directlua{ Babel.locale_props[\the\localeid].textdir = 'r' }%
3865      \or
3866        \directlua{ Babel.locale_props[\the\localeid].textdir = 'al' }%
3867      \fi
3868    \fi}
3869 \def\bbl@switchdir{%
3870    \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
3871    \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
3872    \bbl@exp{\\\bbl@setdirs\bbl@cl{wdir}}}
3873 \def\bbl@setdirs#1{% TODO - math
3874    \ifcase\bbl@select@type % TODO - strictly, not the right test
3875      \bbl@bodydir{#1}%
3876      \bbl@pardir{#1}%
3877    \fi
3878    \bbl@textdir{#1}}
3879 % TODO. Only if \bbl@bidimode > 0?:
3880 \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
3881 \DisableBabelHook{babel-bidi}
```

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

```
3882 \ifodd\bbl@engine  % luatex=1
3883 \else % pdftex=0, xetex=2
3884    \newcount\bbl@dirlevel
3885    \chardef\bbl@thetextdir\z@
3886    \chardef\bbl@thepardir\z@
3887    \def\bbl@textdir#1{%
3888      \ifcase#1\relax
3889        \chardef\bbl@thetextdir\z@
3890        \bbl@textdir@i\beginL\endL
```

```
3891        \else
3892          \chardef\bbl@thetextdir\@ne
3893          \bbl@textdir@i\beginR\endR
3894      \fi}
3895    \def\bbl@textdir@i#1#2{%
3896      \ifhmode
3897        \ifnum\currentgrouplevel>\z@
3898          \ifnum\currentgrouplevel=\bbl@dirlevel
3899            \bbl@error{Multiple bidi settings inside a group}%
3900              {I'll insert a new group, but expect wrong results.}%
3901            \bgroup\aftergroup#2\aftergroup\egroup
3902          \else
3903            \ifcase\currentgrouptype\or % 0 bottom
3904              \aftergroup#2% 1 simple {}
3905            \or
3906              \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
3907            \or
3908              \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
3909            \or\or\or % vbox vtop align
3910            \or
3911              \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
3912            \or\or\or\or\or\or % output math disc insert vcent mathchoice
3913            \or
3914              \aftergroup#2% 14 \begingroup
3915            \else
3916              \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
3917            \fi
3918          \fi
3919          \bbl@dirlevel\currentgrouplevel
3920        \fi
3921        #1%
3922      \fi}
3923    \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
3924    \let\bbl@bodydir\@gobble
3925    \let\bbl@pagedir\@gobble
3926    \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).

```
3927    \def\bbl@xebidipar{%
3928      \let\bbl@xebidipar\relax
3929      \TeXXeTstate\@ne
3930      \def\bbl@xeeverypar{%
3931        \ifcase\bbl@thepardir
3932          \ifcase\bbl@thetextdir\else\beginR\fi
3933        \else
3934          {\setbox\z@\lastbox\beginR\box\z@}%
3935        \fi}%
3936      \let\bbl@severypar\everypar
3937      \newtoks\everypar
3938      \everypar=\bbl@severypar
3939      \bbl@severypar{\bbl@xeeverypar\the\everypar}}
3940    \ifnum\bbl@bidimode>200
3941      \let\bbl@textdir@i\@gobbletwo
3942      \let\bbl@xebidipar\@empty
3943      \AddBabelHook{bidi}{foreign}{%
3944        \def\bbl@tempa{\def\BabelText####1}%
3945        \ifcase\bbl@thetextdir
```

```
3946        \expandafter\bbl@tempa\expandafter{\BabelText{\LR{##1}}}%
3947      \else
3948        \expandafter\bbl@tempa\expandafter{\BabelText{\RL{##1}}}%
3949      \fi}
3950    \def\bbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}
3951  \fi
3952 \fi
```

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

```
3953 \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}
3954 \AtBeginDocument{%
3955  \ifx\pdfstringdefDisableCommands\@undefined\else
3956   \ifx\pdfstringdefDisableCommands\relax\else
3957     \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
3958   \fi
3959  \fi}
```

9.6   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.

```
3960 \bbl@trace{Local Language Configuration}
3961 \ifx\loadlocalcfg\@undefined
3962  \@ifpackagewith{babel}{noconfigs}%
3963    {\let\loadlocalcfg\@gobble}%
3964    {\def\loadlocalcfg#1{%
3965      \InputIfFileExists{#1.cfg}%
3966        {\typeout{*************************************^^J%
3967                     * Local config file #1.cfg used^^J%
3968                     *}}%
3969        \@empty}}
3970 \fi
```

9.7   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).

```
3971 \bbl@trace{Language options}
3972 \let\bbl@afterlang\relax
3973 \let\BabelModifiers\relax
3974 \let\bbl@loaded\@empty
3975 \def\bbl@load@language#1{%
3976  \InputIfFileExists{#1.ldf}%
3977    {\edef\bbl@loaded{\CurrentOption
3978       \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}%
3979     \expandafter\let\expandafter\bbl@afterlang
3980       \csname\CurrentOption.ldf-h@@k\endcsname
3981     \expandafter\let\expandafter\BabelModifiers
3982       \csname bbl@mod@\CurrentOption\endcsname}%
3983    {\bbl@error{%
3984      Unknown option '\CurrentOption'. Either you misspelled it\\%
3985      or the language definition file \CurrentOption.ldf was not found}{%
3986      Valid options are, among others: shorthands=, KeepShorthandsActive,\\%
3987      activeacute, activegrave, noconfigs, safe=, main=, math=\\%
```

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.

```
3989 \def\bbl@try@load@lang#1#2#3{%
3990   \IfFileExists{\CurrentOption.ldf}%
3991     {\bbl@load@language{\CurrentOption}}%
3992     {#1\bbl@load@language{#2}#3}}
3993 %
3994 \DeclareOption{hebrew}{%
3995   \input{rlbabel.def}%
3996   \bbl@load@language{hebrew}}
3997 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}{}}
3998 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}{}}
3999 \DeclareOption{nynorsk}{\bbl@try@load@lang{}{norsk}{}}
4000 \DeclareOption{polutonikogreek}{%
4001   \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}
4002 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}{}}
4003 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}{}}
4004 \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.

```
4005 \ifx\bbl@opt@config\@nnil
4006   \@ifpackagewith{babel}{noconfigs}{}%
4007     {\InputIfFileExists{bblopts.cfg}%
4008       {\typeout{*************************************^^J%
4009               * Local config file bblopts.cfg used^^J%
4010               *}}%
4011       {}}%
4012 \else
4013   \InputIfFileExists{\bbl@opt@config.cfg}%
4014     {\typeout{*************************************^^J%
4015             * Local config file \bbl@opt@config.cfg used^^J%
4016             *}}%
4017     {\bbl@error{%
4018       Local config file '\bbl@opt@config.cfg' not found}{%
4019       Perhaps you misspelled it.}}%
4020 \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 (note this list also contains the language given with main). If not declared above, the names of the option and the file are the same.

```
4021 \let\bbl@tempc\relax
4022 \bbl@foreach\bbl@language@opts{%
4023   \ifcase\bbl@iniflag  % Default
4024     \bbl@ifunset{ds@#1}%
4025       {\DeclareOption{#1}{\bbl@load@language{#1}}}%
4026       {}%
4027   \or    % provide=*
4028     \@gobble % case 2 same as 1
4029   \or    % provide+=*
4030     \bbl@ifunset{ds@#1}%
4031       {\IfFileExists{#1.ldf}{}%
4032         {\IfFileExists{babel-#1.tex}{}{\@namedef{ds@#1}{}}}}%
```

```
4033          {}%
4034     \bbl@ifunset{ds@#1}%
4035        {\def\bbl@tempc{#1}%
4036         \DeclareOption{#1}{%
4037           \ifnum\bbl@iniflag>\@ne
4038             \bbl@ldfinit
4039             \babelprovide[import]{#1}%
4040             \bbl@afterldf{}%
4041           \else
4042             \bbl@load@language{#1}%
4043           \fi}}%
4044        {}%
4045   \or     % provide*=*
4046     \def\bbl@tempc{#1}%
4047     \bbl@ifunset{ds@#1}%
4048        {\DeclareOption{#1}{%
4049           \bbl@ldfinit
4050           \babelprovide[import]{#1}%
4051           \bbl@afterldf{}}}%
4052        {}%
4053   \fi}
```

Now, we make sure an option is explicitly declared for any language set as global option, by checking if an ldf exists. The previous step was, in fact, somewhat redundant, but that way we minimize accessing the file system just to see if the option could be a language.

```
4054 \let\bbl@tempb\@nnil
4055 \let\bbl@clsoptlst\@classoptionslist
4056 \bbl@foreach\@classoptionslist{%
4057   \bbl@ifunset{ds@#1}%
4058      {\IfFileExists{#1.ldf}%
4059        {\def\bbl@tempb{#1}%
4060         \DeclareOption{#1}{%
4061           \ifnum\bbl@iniflag>\@ne
4062             \bbl@ldfinit
4063             \babelprovide[import]{#1}%
4064             \bbl@afterldf{}%
4065           \else
4066             \bbl@load@language{#1}%
4067           \fi}}%
4068        {\IfFileExists{babel-#1.tex}%
4069          {\def\bbl@tempb{#1}%
4070           \ifnum\bbl@iniflag>\z@
4071             \DeclareOption{#1}{%
4072               \ifnum\bbl@iniflag>\@ne
4073                 \bbl@ldfinit
4074                 \babelprovide[import]{#1}%
4075                 \bbl@afterldf{}%
4076               \fi}%
4077           \fi}%
4078          {}}}%
4079      {}}
```

If a main language has been set, store it for the third pass.

```
4080 \ifnum\bbl@iniflag=\z@\else
4081   \ifx\bbl@opt@main\@nnil
4082     \ifx\bbl@tempc\relax
4083       \let\bbl@opt@main\bbl@tempb
4084     \else
4085       \let\bbl@opt@main\bbl@tempc
```

```
4086     \fi
4087   \fi
4088 \fi
4089 \ifx\bbl@opt@main\@nnil\else
4090   \expandafter
4091   \let\expandafter\bbl@loadmain\csname ds@\bbl@opt@main\endcsname
4092   \expandafter\let\csname ds@\bbl@opt@main\endcsname\@empty
4093 \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 (except, of course, global options, which LaTeX processes before):

```
4094 \def\AfterBabelLanguage#1{%
4095   \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang}{}}
4096 \DeclareOption*{}
4097 \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. Then execute directly the option (because it could be used only in main). After loading all languages, we deactivate \AfterBabelLanguage.

```
4098 \bbl@trace{Option 'main'}
4099 \ifx\bbl@opt@main\@nnil
4100   \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}
4101   \let\bbl@tempc\@empty
4102   \bbl@for\bbl@tempb\bbl@tempa{%
4103     \bbl@xin@{,\bbl@tempb,}{,\bbl@loaded,}%
4104     \ifin@\edef\bbl@tempc{\bbl@tempb}\fi}
4105   \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
4106   \expandafter\bbl@tempa\bbl@loaded,\@nnil
4107   \ifx\bbl@tempb\bbl@tempc\else
4108     \bbl@warning{%
4109       Last declared language option is '\bbl@tempc',\\%
4110       but the last processed one was '\bbl@tempb'.\\%
4111       The main language can't be set as both a global\\%
4112       and a package option. Use 'main=\bbl@tempc' as\\%
4113       option. Reported}%
4114   \fi
4115 \else
4116   \ifodd\bbl@iniflag  % case 1,3
4117     \bbl@ldfinit
4118     \let\CurrentOption\bbl@opt@main
4119     \ifx\bbl@opt@provide\@nnil
4120       \bbl@exp{\\\babelprovide[import,main]{\bbl@opt@main}}%
4121     \else
4122       \bbl@exp{\\\bbl@forkv{\@nameuse{@raw@opt@babel.sty}}}{%
4123         \bbl@xin@{,provide,}{,#1,}%
4124         \ifin@
4125           \def\bbl@opt@provide{#2}%
4126           \bbl@replace\bbl@opt@provide{;}{,}%
4127         \fi}%
4128       \bbl@exp{%
4129         \\\babelprovide[\bbl@opt@provide,import,main]{\bbl@opt@main}}%
4130     \fi
4131     \bbl@afterldf{}%
4132   \else % case 0,2
4133     \chardef\bbl@iniflag\z@  % Force ldf
4134     \expandafter\let\csname ds@\bbl@opt@main\endcsname\bbl@loadmain
```

```
4135      \ExecuteOptions{\bbl@opt@main}
4136      \DeclareOption*{}%
4137      \ProcessOptions*
4138   \fi
4139 \fi
4140 \def\AfterBabelLanguage{%
4141    \bbl@error
4142      {Too late for \string\AfterBabelLanguage}%
4143      {Languages have been loaded, so I can do nothing}}
```

In order to catch the case where the user forgot to specify a language we check whether \bbl@main@language, has become defined. If not, no language has been loaded and an error message is displayed.

```
4144 \ifx\bbl@main@language\@undefined
4145    \bbl@info{%
4146      You haven't specified a language. I'll use 'nil'\\%
4147      as the main language. Reported}
4148      \bbl@load@language{nil}
4149 \fi
4150 ⟨/package⟩
```

10   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

```
4151 ⟨*kernel⟩
4152 \let\bbl@onlyswitch\@empty
4153 \input babel.def
4154 \let\bbl@onlyswitch\@undefined
4155 ⟨/kernel⟩
4156 ⟨*patterns⟩
```

11   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.

```
4157 ⟨⟨Make sure ProvidesFile is defined⟩⟩
4158 \ProvidesFile{hyphen.cfg}[⟨⟨date⟩⟩ ⟨⟨version⟩⟩ Babel hyphens]
4159 \xdef\bbl@format{\jobname}
4160 \def\bbl@version{⟨⟨version⟩⟩}
4161 \def\bbl@date{⟨⟨date⟩⟩}
4162 \ifx\AtBeginDocument\@undefined
4163    \def\@empty{}
4164 \fi
4165 ⟨⟨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.

```
4166 \def\process@line#1#2 #3 #4 {%
4167   \ifx=#1%
4168     \process@synonym{#2}%
4169   \else
4170     \process@language{#1#2}{#3}{#4}%
4171   \fi
4172   \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.

```
4173 \toks@{}
4174 \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.

```
4175 \def\process@synonym#1{%
4176   \ifnum\last@language=\m@ne
4177     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4178   \else
4179     \expandafter\chardef\csname l@#1\endcsname\last@language
4180     \wlog{\string\l@#1=\string\language\the\last@language}%
4181     \expandafter\let\csname #1hyphenmins\expandafter\endcsname
4182       \csname\languagename hyphenmins\endcsname
4183     \let\bbl@elt\relax
4184     \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}{}{}}%
4185   \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.

157

Finally, if the counter \language is equal to zero we execute the synonyms stored.

```
4186 \def\process@language#1#2#3{%
4187   \expandafter\addlanguage\csname l@#1\endcsname
4188   \expandafter\language\csname l@#1\endcsname
4189   \edef\languagename{#1}%
4190   \bbl@hook@everylanguage{#1}%
4191   % > luatex
4192   \bbl@get@enc#1::\@@@
4193   \begingroup
4194     \lefthyphenmin\m@ne
4195     \bbl@hook@loadpatterns{#2}%
4196     % > luatex
4197     \ifnum\lefthyphenmin=\m@ne
4198     \else
4199       \expandafter\xdef\csname #1hyphenmins\endcsname{%
4200         \the\lefthyphenmin\the\righthyphenmin}%
4201     \fi
4202   \endgroup
4203   \def\bbl@tempa{#3}%
4204   \ifx\bbl@tempa\@empty\else
4205     \bbl@hook@loadexceptions{#3}%
4206     % > luatex
4207   \fi
4208   \let\bbl@elt\relax
4209   \edef\bbl@languages{%
4210     \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
4211   \ifnum\the\language=\z@
4212     \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4213       \set@hyphenmins\tw@\thr@@\relax
4214     \else
4215       \expandafter\expandafter\expandafter\set@hyphenmins
4216         \csname #1hyphenmins\endcsname
4217     \fi
4218     \the\toks@
4219     \toks@{}%
4220   \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.

```
4221 \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.

```
4222 \def\bbl@hook@everylanguage#1{}
4223 \def\bbl@hook@loadpatterns#1{\input #1\relax}
4224 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
4225 \def\bbl@hook@loadkernel#1{%
4226   \def\addlanguage{\csname newlanguage\endcsname}%
4227   \def\adddialect##1##2{%
4228     \global\chardef##1##2\relax
4229     \wlog{\string##1 = a dialect from \string\language##2}}%
4230   \def\iflanguage##1{%
4231     \expandafter\ifx\csname l@##1\endcsname\relax
4232       \@nolanerr{##1}%
4233     \else
4234       \ifnum\csname l@##1\endcsname=\language
4235         \expandafter\expandafter\expandafter\@firstoftwo
```

```
4236        \else
4237          \expandafter\expandafter\expandafter\@secondoftwo
4238        \fi
4239      \fi}%
4240    \def\providehyphenmins##1##2{%
4241      \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
4242        \@namedef{##1hyphenmins}{##2}%
4243      \fi}%
4244    \def\set@hyphenmins##1##2{%
4245      \lefthyphenmin##1\relax
4246      \righthyphenmin##2\relax}%
4247    \def\selectlanguage{%
4248      \errhelp{Selecting a language requires a package supporting it}%
4249      \errmessage{Not loaded}}%
4250    \let\foreignlanguage\selectlanguage
4251    \let\otherlanguage\selectlanguage
4252    \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4253    \def\bbl@usehooks##1##2{}% TODO. Temporary!!
4254    \def\setlocale{%
4255      \errhelp{Find an armchair, sit down and wait}%
4256      \errmessage{Not yet available}}%
4257    \let\uselocale\setlocale
4258    \let\locale\setlocale
4259    \let\selectlocale\setlocale
4260    \let\localename\setlocale
4261    \let\textlocale\setlocale
4262    \let\textlanguage\setlocale
4263    \let\languagetext\setlocale}
4264 \begingroup
4265    \def\AddBabelHook#1#2{%
4266      \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
4267        \def\next{\toks1}%
4268      \else
4269        \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname####1}%
4270      \fi
4271      \next}
4272    \ifx\directlua\@undefined
4273      \ifx\XeTeXinputencoding\@undefined\else
4274        \input xebabel.def
4275      \fi
4276    \else
4277      \input luababel.def
4278    \fi
4279    \openin1 = babel-\bbl@format.cfg
4280    \ifeof1
4281    \else
4282      \input babel-\bbl@format.cfg\relax
4283    \fi
4284    \closein1
4285 \endgroup
4286 \bbl@hook@loadkernel{switch.def}
```

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

```
4287 \openin1 = language.dat
```

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

```
4288 \def\languagename{english}%
4289 \ifeof1
```

```
4290   \message{I couldn't find the file language.dat,\space
4291            I will try the file hyphen.tex}
4292   \input hyphen.tex\relax
4293   \chardef\l@english\z@
4294 \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.

```
4295   \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.

```
4296   \loop
4297     \endlinechar\m@ne
4298     \read1 to \bbl@line
4299     \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.

```
4300     \if T\ifeof1F\fi T\relax
4301       \ifx\bbl@line\@empty\else
4302         \edef\bbl@line{\bbl@line\space\space\space}%
4303         \expandafter\process@line\bbl@line\relax
4304       \fi
4305   \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.

```
4306   \begingroup
4307     \def\bbl@elt#1#2#3#4{%
4308       \global\language=#2\relax
4309       \gdef\languagename{#1}%
4310       \def\bbl@elt##1##2##3##4{}}%
4311     \bbl@languages
4312   \endgroup
4313 \fi
4314 \closein1
```

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

```
4315 \if/\the\toks@/\else
4316   \errhelp{language.dat loads no language, only synonyms}
4317   \errmessage{Orphan language synonym}
4318 \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.

```
4319 \let\bbl@line\@undefined
4320 \let\process@line\@undefined
4321 \let\process@synonym\@undefined
4322 \let\process@language\@undefined
4323 \let\bbl@get@enc\@undefined
4324 \let\bbl@hyph@enc\@undefined
4325 \let\bbl@tempa\@undefined
4326 \let\bbl@hook@loadkernel\@undefined
4327 \let\bbl@hook@everylanguage\@undefined
```

160

```
4328 \let\bbl@hook@loadpatterns\@undefined
4329 \let\bbl@hook@loadexceptions\@undefined
4330 ⟨/patterns⟩
```

Here the code for iniTeX ends.

12   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].

```
4331 ⟨⟨*More package options⟩⟩ ≡
4332 \chardef\bbl@bidimode\z@
4333 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4334 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
4335 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4336 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4337 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4338 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4339 ⟨⟨/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.

```
4340 ⟨⟨*Font selection⟩⟩ ≡
4341 \bbl@trace{Font handling with fontspec}
4342 \ifx\ExplSyntaxOn\@undefined\else
4343  \ExplSyntaxOn
4344  \catcode`\ =10
4345  \def\bbl@loadfontspec{%
4346    \usepackage{fontspec}%  TODO. Apply patch always
4347    \expandafter
4348    \def\csname msg~text~>~fontspec/language-not-exist\endcsname##1##2##3##4{%
4349      Font '\l_fontspec_fontname_tl' is using the\\%
4350      default features for language '##1'.\\%
4351      That's usually fine, because many languages\\%
4352      require no specific features, but if the output is\\%
4353      not as expected, consider selecting another font.}
4354    \expandafter
4355    \def\csname msg~text~>~fontspec/no-script\endcsname##1##2##3##4{%
4356      Font '\l_fontspec_fontname_tl' is using the\\%
4357      default features for script '##2'.\\%
4358      That's not always wrong, but if the output is\\%
4359      not as expected, consider selecting another font.}}
4360  \ExplSyntaxOff
4361 \fi
4362 \@onlypreamble\babelfont
4363 \newcommand\babelfont[2][]{%  1=langs/scripts 2=fam
4364  \bbl@foreach{#1}{%
4365    \expandafter\ifx\csname date##1\endcsname\relax
4366      \IfFileExists{babel-##1.tex}%
4367        {\babelprovide{##1}}%
4368        {}%
4369    \fi}%
4370  \edef\bbl@tempa{#1}%
4371  \def\bbl@tempb{#2}%  Used by \bbl@bblfont
```

```
4372    \ifx\fontspec\@undefined
4373      \bbl@loadfontspec
4374    \fi
4375    \EnableBabelHook{babel-fontspec}% Just calls \bbl@switchfont
4376    \bbl@bblfont}
4377 \newcommand\bbl@bblfont[2][]{% 1=features 2=fontname, @font=rm|sf|tt
4378    \bbl@ifunset{\bbl@tempb family}%
4379      {\bbl@providefam{\bbl@tempb}}%
4380      {}%
4381    % For the default font, just in case:
4382    \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
4383    \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
4384      {\bbl@csarg\edef{\bbl@tempb dflt@}{<>{#1}{#2}}% save bbl@rmdflt@
4385       \bbl@exp{%
4386         \let\<bbl@\bbl@tempb dflt@\languagename>\<bbl@\bbl@tempb dflt@>%
4387         \\\bbl@font@set\<bbl@\bbl@tempb dflt@\languagename>%
4388                       \<\bbl@tempb default>\<\bbl@tempb family>}}%
4389      {\bbl@foreach\bbl@tempa{% ie bbl@rmdflt@lang / *scrt
4390         \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:

```
4391 \def\bbl@providefam#1{%
4392    \bbl@exp{%
4393      \\\newcommand\<#1default>{}% Just define it
4394      \\\bbl@add@list\\\bbl@font@fams{#1}%
4395      \\\DeclareRobustCommand\<#1family>{%
4396        \\\not@math@alphabet\<#1family>\relax
4397        % \\\prepare@family@series@update{#1}\<#1default>% TODO. Fails
4398        \\\fontfamily\<#1default>%
4399        \<ifx>\\\UseHooks\\\@undefined\<else>\\\UseHook{#1family}\<fi>%
4400        \\\selectfont}%
4401      \\\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.

```
4402 \def\bbl@nostdfont#1{%
4403    \bbl@ifunset{bbl@WFF@\f@family}%
4404      {\bbl@csarg\gdef{WFF@\f@family}{}%  Flag, to avoid dupl warns
4405       \bbl@infowarn{The current font is not a babel standard family:\\%
4406         #1%
4407         \fontname\font\\%
4408         There is nothing intrinsically wrong with this warning, and\\%
4409         you can ignore it altogether if you do not need these\\%
4410         families. But if they are used in the document, you should be\\%
4411         aware 'babel' will no set Script and Language for them, so\\%
4412         you may consider defining a new family with \string\babelfont.\\%
4413         See the manual for further details about \string\babelfont.\\%
4414         Reported}}
4415      {}}%
4416 \gdef\bbl@switchfont{%
4417    \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
4418    \bbl@exp{%  eg Arabic -> arabic
4419      \lowercase{\edef\\\bbl@tempa{\bbl@cl{sname}}}}%
4420    \bbl@foreach\bbl@font@fams{%
4421      \bbl@ifunset{bbl@##1dflt@\languagename}%     (1) language?
4422        {\bbl@ifunset{bbl@##1dflt@*\bbl@tempa}%     (2) from script?
4423          {\bbl@ifunset{bbl@##1dflt@}%              2=F - (3) from generic?
4424            {}%                                     123=F - nothing!
4425            {\bbl@exp{%                             3=T - from generic
```

```
4426            \global\let\<bbl@##1dflt@\languagename>%
4427                         \<bbl@##1dflt@>}}}%
4428         {\bbl@exp{%                              2=T - from script
4429             \global\let\<bbl@##1dflt@\languagename>%
4430                         \<bbl@##1dflt@*\bbl@tempa>}}}%
4431     {}}%                                        1=T - language, already defined
4432 \def\bbl@tempa{\bbl@nostdfont{}}%
4433 \bbl@foreach\bbl@font@fams{%    don't gather with prev for
4434    \bbl@ifunset{bbl@##1dflt@\languagename}%
4435      {\bbl@cs{famrst@##1}%
4436       \global\bbl@csarg\let{famrst@##1}\relax}%
4437      {\bbl@exp{% order is relevant. TODO: but sometimes wrong!
4438         \\\bbl@add\\\originalTeX{%
4439           \\\bbl@font@rst{\bbl@cl{##1dflt}}%
4440                         \<##1default>\<##1family>{##1}}%
4441         \\\bbl@font@set\<bbl@##1dflt@\languagename>% the main part!
4442                         \<##1default>\<##1family>}}}%
4443    \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.

```
4444 \ifx\f@family\@undefined\else    % if latex
4445   \ifcase\bbl@engine              % if pdftex
4446     \let\bbl@ckeckstdfonts\relax
4447   \else
4448     \def\bbl@ckeckstdfonts{%
4449       \begingroup
4450         \global\let\bbl@ckeckstdfonts\relax
4451         \let\bbl@tempa\@empty
4452         \bbl@foreach\bbl@font@fams{%
4453           \bbl@ifunset{bbl@##1dflt@}%
4454             {\@nameuse{##1family}%
4455              \bbl@csarg\gdef{WFF@\f@family}{}% Flag
4456              \bbl@exp{\\\bbl@add\\\bbl@tempa{* \<##1family>= \f@family\\\\%
4457                 \space\space\fontname\font\\\\}}%
4458              \bbl@csarg\xdef{##1dflt@}{\f@family}%
4459              \expandafter\xdef\csname ##1default\endcsname{\f@family}}%
4460             {}}%
4461         \ifx\bbl@tempa\@empty\else
4462           \bbl@infowarn{The following font families will use the default\\%
4463             settings for all or some languages:\\%
4464             \bbl@tempa
4465             There is nothing intrinsically wrong with it, but\\%
4466             'babel' will no set Script and Language, which could\\%
4467              be relevant in some languages. If your document uses\\%
4468              these families, consider redefining them with \string\babelfont.\\%
4469             Reported}%
4470         \fi
4471       \endgroup}
4472   \fi
4473 \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.

```
4474 \def\bbl@font@set#1#2#3{% eg \bbl@rmdflt@lang \rmdefault \rmfamily
4475   \bbl@xin@{<>}{#1}%
4476   \ifin@
```

```
4477    \bbl@exp{\\\bbl@fontspec@set\\#1\expandafter\@gobbletwo#1\\#3}%
4478  \fi
4479  \bbl@exp{%                'Unprotected' macros return prev values
4480    \def\\#2{#1}%           eg, \rmdefault{\bbl@rmdflt@lang}
4481    \\\bbl@ifsamestring{#2}{\f@family}%
4482      {\\#3%
4483       \\\bbl@ifsamestring{\f@series}{\bfdefault}{\\\bfseries}{}%
4484       \let\\\bbl@tempa\relax}%
4485      {}}}
4486 %    TODO - next should be global?, but even local does its job. I'm
4487 %    still not sure -- must investigate:
4488 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
4489    \let\bbl@tempe\bbl@mapselect
4490    \let\bbl@mapselect\relax
4491    \let\bbl@temp@fam#4%         eg, '\rmfamily', to be restored below
4492    \let#4\@empty       %        Make sure \renewfontfamily is valid
4493    \bbl@exp{%
4494      \let\\\bbl@temp@pfam\<\bbl@stripslash#4\space>%  eg, '\rmfamily '
4495      \<keys_if_exist:nnF>{fontspec-opentype}{Script/\bbl@cl{sname}}%
4496        {\\\newfontscript{\bbl@cl{sname}}{\bbl@cl{sotf}}}%
4497      \<keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@cl{lname}}%
4498        {\\\newfontlanguage{\bbl@cl{lname}}{\bbl@cl{lotf}}}%
4499      \\\renewfontfamily\\#4%
4500        [\bbl@cl{lsys},#2]}{#3}% ie \bbl@exp{..}{#3}
4501    \begingroup
4502      #4%
4503      \xdef#1{\f@family}%     eg, \bbl@rmdflt@lang{FreeSerif(0)}
4504    \endgroup
4505    \let#4\bbl@temp@fam
4506    \bbl@exp{\let\<\bbl@stripslash#4\space>}\bbl@temp@pfam
4507    \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.

```
4508 \def\bbl@font@rst#1#2#3#4{%
4509    \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.

```
4510 \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 :-).

```
4511 \newcommand\babelFSstore[2][]{%
4512    \bbl@ifblank{#1}%
4513      {\bbl@csarg\def{sname@#2}{Latin}}%
4514      {\bbl@csarg\def{sname@#2}{#1}}%
4515    \bbl@provide@dirs{#2}%
4516    \bbl@csarg\ifnum{wdir@#2}>\z@
4517      \let\bbl@beforeforeign\leavevmode
4518      \EnableBabelHook{babel-bidi}%
4519    \fi
4520    \bbl@foreach{#2}{%
4521      \bbl@FSstore{##1}{rm}\rmdefault\bbl@save@rmdefault
4522      \bbl@FSstore{##1}{sf}\sfdefault\bbl@save@sfdefault
4523      \bbl@FSstore{##1}{tt}\ttdefault\bbl@save@ttdefault}}
4524 \def\bbl@FSstore#1#2#3#4{%
4525    \bbl@csarg\edef{#2default#1}{#3}%
4526    \expandafter\addto\csname extras#1\endcsname{%
```

```
4527        \let#4#3%
4528        \ifx#3\f@family
4529          \edef#3{\csname bbl@#2default#1\endcsname}%
4530          \fontfamily{#3}\selectfont
4531        \else
4532          \edef#3{\csname bbl@#2default#1\endcsname}%
4533        \fi}%
4534      \expandafter\addto\csname noextras#1\endcsname{%
4535        \ifx#3\f@family
4536          \fontfamily{#4}\selectfont
4537        \fi
4538        \let#3#4}}
4539    \let\bbl@langfeatures\@empty
4540    \def\babelFSfeatures{% make sure \fontspec is redefined once
4541      \let\bbl@ori@fontspec\fontspec
4542      \renewcommand\fontspec[1][]{%
4543        \bbl@ori@fontspec[\bbl@langfeatures##1]}
4544      \let\babelFSfeatures\bbl@FSfeatures
4545      \babelFSfeatures}
4546    \def\bbl@FSfeatures#1#2{%
4547      \expandafter\addto\csname extras#1\endcsname{%
4548        \babel@save\bbl@langfeatures
4549        \edef\bbl@langfeatures{#2,}}}
4550  ⟨⟨/Font selection⟩⟩
```

13 Hooks for XeTeX and LuaTeX

13.1 XeTeX

Unfortunately, the current encoding cannot be retrieved and therefore it is reset always to utf8,
which seems a sensible default.

```
4551  ⟨⟨∗Footnote changes⟩⟩ ≡
4552  \bbl@trace{Bidi footnotes}
4553  \ifnum\bbl@bidimode>\z@
4554    \def\bbl@footnote#1#2#3{%
4555      \@ifnextchar[%
4556        {\bbl@footnote@o{#1}{#2}{#3}}%
4557        {\bbl@footnote@x{#1}{#2}{#3}}}
4558    \long\def\bbl@footnote@x#1#2#3#4{%
4559      \bgroup
4560        \select@language@x{\bbl@main@language}%
4561        \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
4562      \egroup}
4563    \long\def\bbl@footnote@o#1#2#3[#4]#5{%
4564      \bgroup
4565        \select@language@x{\bbl@main@language}%
4566        \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
4567      \egroup}
4568    \def\bbl@footnotetext#1#2#3{%
4569      \@ifnextchar[%
4570        {\bbl@footnotetext@o{#1}{#2}{#3}}%
4571        {\bbl@footnotetext@x{#1}{#2}{#3}}}
4572    \long\def\bbl@footnotetext@x#1#2#3#4{%
4573      \bgroup
4574        \select@language@x{\bbl@main@language}%
4575        \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
4576      \egroup}
4577    \long\def\bbl@footnotetext@o#1#2#3[#4]#5{%
```

```
4578        \bgroup
4579          \select@language@x{\bbl@main@language}%
4580          \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
4581        \egroup}
4582    \def\BabelFootnote#1#2#3#4{%
4583      \ifx\bbl@fn@footnote\@undefined
4584        \let\bbl@fn@footnote\footnote
4585      \fi
4586      \ifx\bbl@fn@footnotetext\@undefined
4587        \let\bbl@fn@footnotetext\footnotetext
4588      \fi
4589      \bbl@ifblank{#2}%
4590        {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
4591         \@namedef{\bbl@stripslash#1text}%
4592           {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
4593        {\def#1{\bbl@exp{\\\bbl@footnote{\\\foreignlanguage{#2}}}{#3}{#4}}%
4594         \@namedef{\bbl@stripslash#1text}%
4595           {\bbl@exp{\\\bbl@footnotetext{\\\foreignlanguage{#2}}}{#3}{#4}}}}
4596  \fi
4597  ⟨⟨/Footnote changes⟩⟩
```

Now, the code.

```
4598  ⟨∗xetex⟩
4599  \def\BabelStringsDefault{unicode}
4600  \let\xebbl@stop\relax
4601  \AddBabelHook{xetex}{encodedcommands}{%
4602    \def\bbl@tempa{#1}%
4603    \ifx\bbl@tempa\@empty
4604      \XeTeXinputencoding"bytes"%
4605    \else
4606      \XeTeXinputencoding"#1"%
4607    \fi
4608    \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4609  \AddBabelHook{xetex}{stopcommands}{%
4610    \xebbl@stop
4611    \let\xebbl@stop\relax}
4612  \def\bbl@intraspace#1 #2 #3\@@{%
4613    \bbl@csarg\gdef{xeisp@\languagename}%
4614      {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4615  \def\bbl@intrapenalty#1\@@{%
4616    \bbl@csarg\gdef{xeipn@\languagename}%
4617      {\XeTeXlinebreakpenalty #1\relax}}
4618  \def\bbl@provide@intraspace{%
4619    \bbl@xin@{/s}{/\bbl@cl{lnbrk}}%
4620    \ifin@\else\bbl@xin@{/c}{/\bbl@cl{lnbrk}}\fi
4621    \ifin@
4622      \bbl@ifunset{bbl@intsp@\languagename}{}%
4623        {\expandafter\ifx\csname bbl@intsp@\languagename\endcsname\@empty\else
4624          \ifx\bbl@KVP@intraspace\@nil
4625            \bbl@exp{%
4626              \\\bbl@intraspace\bbl@cl{intsp}\\\@@}%
4627          \fi
4628          \ifx\bbl@KVP@intrapenalty\@nil
4629            \bbl@intrapenalty0\@@
4630          \fi
4631        \fi
4632        \ifx\bbl@KVP@intraspace\@nil\else % We may override the ini
4633          \expandafter\bbl@intraspace\bbl@KVP@intraspace\@@
4634        \fi
```

166

```
4635        \ifx\bbl@KVP@intrapenalty\@nil\else
4636          \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
4637        \fi
4638        \bbl@exp{%
4639          % TODO. Execute only once (but redundant):
4640          \\\bbl@add\<extras\languagename>{%
4641            \XeTeXlinebreaklocale "\bbl@cl{tbcp}"%
4642            \<bbl@xeisp@\languagename>%
4643            \<bbl@xeipn@\languagename>}%
4644          \\\bbl@toglobal\<extras\languagename>%
4645          \\\bbl@add\<noextras\languagename>{%
4646            \XeTeXlinebreaklocale "en"}%
4647          \\\bbl@toglobal\<noextras\languagename>}%
4648        \ifx\bbl@ispacesize\@undefined
4649          \gdef\bbl@ispacesize{\bbl@cl{xeisp}}%
4650          \ifx\AtBeginDocument\@notprerr
4651            \expandafter\@secondoftwo  % to execute right now
4652          \fi
4653          \AtBeginDocument{\bbl@patchfont{\bbl@ispacesize}}%
4654        \fi}%
4655  \fi}
4656 \ifx\DisableBabelHook\@undefined\endinput\fi
4657 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4658 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
4659 \DisableBabelHook{babel-fontspec}
4660 ⟨⟨Font selection⟩⟩
4661 \input txtbabel.def
4662 ⟨/xetex⟩
```

13.2 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.

```
4663 ⟨*texxet⟩
4664 \providecommand\bbl@provide@intraspace{}
4665 \bbl@trace{Redefinitions for bidi layout}
4666 \def\bbl@sspre@caption{%
4667   \bbl@exp{\everyhbox{\\\bbl@textdir\bbl@cs{wdir@\bbl@main@language}}}}
4668 \ifx\bbl@opt@layout\@nnil\endinput\fi  % No layout
4669 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
4670 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
4671 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
4672   \def\@hangfrom#1{%
4673     \setbox\@tempboxa\hbox{{#1}}%
4674     \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
4675     \noindent\box\@tempboxa}
4676   \def\raggedright{%
4677     \let\\\@centercr
4678     \bbl@startskip\z@skip
4679     \@rightskip\@flushglue
4680     \bbl@endskip\@rightskip
4681     \parindent\z@
4682     \parfillskip\bbl@startskip}
```

```
4683    \def\raggedleft{%
4684       \let\\\@centercr
4685       \bbl@startskip\@flushglue
4686       \bbl@endskip\z@skip
4687       \parindent\z@
4688       \parfillskip\bbl@endskip}
4689 \fi
4690 \IfBabelLayout{lists}
4691    {\bbl@sreplace\list
4692       {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
4693    \def\bbl@listleftmargin{%
4694       \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
4695    \ifcase\bbl@engine
4696       \def\labelenumii{)\theenumii(}% pdftex doesn't reverse ()
4697       \def\p@enumiii{\p@enumii)\theenumii(}%
4698    \fi
4699    \bbl@sreplace\@verbatim
4700       {\leftskip\@totalleftmargin}%
4701       {\bbl@startskip\textwidth
4702        \advance\bbl@startskip-\linewidth}%
4703    \bbl@sreplace\@verbatim
4704       {\rightskip\z@skip}%
4705       {\bbl@endskip\z@skip}}%
4706    {}
4707 \IfBabelLayout{contents}
4708    {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
4709     \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
4710    {}
4711 \IfBabelLayout{columns}
4712    {\bbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbl@outputhbox}%
4713    \def\bbl@outputhbox#1{%
4714       \hb@xt@\textwidth{%
4715          \hskip\columnwidth
4716          \hfil
4717          {\normalcolor\vrule \@width\columnseprule}%
4718          \hfil
4719          \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
4720          \hskip-\textwidth
4721          \hb@xt@\columnwidth{\box\@outputbox \hss}%
4722          \hskip\columnsep
4723          \hskip\columnwidth}}}%
4724    {}
4725 ⟨⟨Footnote changes⟩⟩
4726 \IfBabelLayout{footnotes}%
4727    {\BabelFootnote\footnote\languagename{}{}%
4728     \BabelFootnote\localfootnote\languagename{}{}%
4729     \BabelFootnote\mainfootnote{}{}{}}
4730    {}
```

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.

```
4731 \IfBabelLayout{counters}%
4732    {\let\bbl@latinarabic=\@arabic
4733     \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
4734     \let\bbl@asciiroman=\@roman
4735     \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciiroman#1}}}%
4736     \let\bbl@asciiRoman=\@Roman
4737     \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}}{}
4738 ⟨/texxet⟩
```

168

13.3 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`).

```
4739 ⟨*luatex⟩
4740 \ifx\AddBabelHook\@undefined % When plain.def, babel.sty starts
4741 \bbl@trace{Read language.dat}
4742 \ifx\bbl@readstream\@undefined
4743   \csname newread\endcsname\bbl@readstream
4744 \fi
4745 \begingroup
4746   \toks@{}
4747   \count@\z@ % 0=start, 1=0th, 2=normal
4748   \def\bbl@process@line#1#2 #3 #4 {%
4749     \ifx=#1%
4750       \bbl@process@synonym{#2}%
4751     \else
4752       \bbl@process@language{#1#2}{#3}{#4}%
4753     \fi
4754     \ignorespaces}
4755   \def\bbl@manylang{%
4756     \ifnum\bbl@last>\@ne
4757       \bbl@info{Non-standard hyphenation setup}%
4758     \fi
4759     \let\bbl@manylang\relax}
4760   \def\bbl@process@language#1#2#3{%
4761     \ifcase\count@
4762       \@ifundefined{zth@#1}{\count@\tw@}{\count@\@ne}%
4763     \or
```

```
4764        \count@\tw@
4765      \fi
4766      \ifnum\count@=\tw@
4767        \expandafter\addlanguage\csname l@#1\endcsname
4768        \language\allocationnumber
4769        \chardef\bbl@last\allocationnumber
4770        \bbl@manylang
4771        \let\bbl@elt\relax
4772        \xdef\bbl@languages{%
4773          \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
4774      \fi
4775      \the\toks@
4776      \toks@{}}
4777    \def\bbl@process@synonym@aux#1#2{%
4778      \global\expandafter\chardef\csname l@#1\endcsname#2\relax
4779      \let\bbl@elt\relax
4780      \xdef\bbl@languages{%
4781        \bbl@languages\bbl@elt{#1}{#2}{}{}}}%
4782    \def\bbl@process@synonym#1{%
4783      \ifcase\count@
4784        \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
4785      \or
4786        \@ifundefined{zth@#1}{\bbl@process@synonym@aux{#1}{0}}{}%
4787      \else
4788        \bbl@process@synonym@aux{#1}{\the\bbl@last}%
4789      \fi}
4790    \ifx\bbl@languages\@undefined % Just a (sensible?) guess
4791      \chardef\l@english\z@
4792      \chardef\l@USenglish\z@
4793      \chardef\bbl@last\z@
4794      \global\@namedef{bbl@hyphendata@0}{{hyphen.tex}{}}
4795      \gdef\bbl@languages{%
4796        \bbl@elt{english}{0}{hyphen.tex}{}%
4797        \bbl@elt{USenglish}{0}{}{}}
4798    \else
4799      \global\let\bbl@languages@format\bbl@languages
4800      \def\bbl@elt#1#2#3#4{% Remove all except language 0
4801        \ifnum#2>\z@\else
4802          \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
4803        \fi}%
4804      \xdef\bbl@languages{\bbl@languages}%
4805    \fi
4806    \def\bbl@elt#1#2#3#4{\@namedef{zth@#1}{}} % Define flags
4807    \bbl@languages
4808    \openin\bbl@readstream=language.dat
4809    \ifeof\bbl@readstream
4810      \bbl@warning{I couldn't find language.dat. No additional\\%
4811                   patterns loaded. Reported}%
4812    \else
4813      \loop
4814        \endlinechar\m@ne
4815        \read\bbl@readstream to \bbl@line
4816        \endlinechar`\^^M
4817        \if T\ifeof\bbl@readstream F\fi T\relax
4818          \ifx\bbl@line\@empty\else
4819            \edef\bbl@line{\bbl@line\space\space\space}%
4820            \expandafter\bbl@process@line\bbl@line\relax
4821          \fi
4822      \repeat
```

```
4823    \fi
4824 \endgroup
4825 \bbl@trace{Macros for reading patterns files}
4826 \def\bbl@get@enc#1:#2:#3\@@@{\def\bbl@hyph@enc{#2}}
4827 \ifx\babelcatcodetablenum\@undefined
4828   \ifx\newcatcodetable\@undefined
4829     \def\babelcatcodetablenum{5211}
4830     \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4831   \else
4832     \newcatcodetable\babelcatcodetablenum
4833     \newcatcodetable\bbl@pattcodes
4834   \fi
4835 \else
4836   \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4837 \fi
4838 \def\bbl@luapatterns#1#2{%
4839   \bbl@get@enc#1::\@@@
4840   \setbox\z@\hbox\bgroup
4841     \begingroup
4842       \savecatcodetable\babelcatcodetablenum\relax
4843       \initcatcodetable\bbl@pattcodes\relax
4844       \catcodetable\bbl@pattcodes\relax
4845         \catcode`\#=6  \catcode`\$=3 \catcode`\&=4 \catcode`\^=7
4846         \catcode`\_=8  \catcode`\{=1 \catcode`\}=2 \catcode`\~=13
4847         \catcode`\@=11 \catcode`\^^I=10 \catcode`\^^J=12
4848         \catcode`\<=12 \catcode`\>=12 \catcode`\*=12 \catcode`\.=12
4849         \catcode`\-=12 \catcode`\/=12 \catcode`\[=12 \catcode`\]=12
4850         \catcode`\`=12 \catcode`\'=12 \catcode`\"=12
4851         \input #1\relax
4852       \catcodetable\babelcatcodetablenum\relax
4853     \endgroup
4854     \def\bbl@tempa{#2}%
4855     \ifx\bbl@tempa\@empty\else
4856       \input #2\relax
4857     \fi
4858   \egroup}%
4859 \def\bbl@patterns@lua#1{%
4860   \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
4861     \csname l@#1\endcsname
4862     \edef\bbl@tempa{#1}%
4863   \else
4864     \csname l@#1:\f@encoding\endcsname
4865     \edef\bbl@tempa{#1:\f@encoding}%
4866   \fi\relax
4867   \@namedef{lu@texhyphen@loaded@\the\language}{}% Temp
4868   \@ifundefined{bbl@hyphendata@\the\language}%
4869     {\def\bbl@elt##1##2##3##4{%
4870       \ifnum##2=\csname l@\bbl@tempa\endcsname % #2=spanish, dutch:OT1...
4871         \def\bbl@tempb{##3}%
4872         \ifx\bbl@tempb\@empty\else % if not a synonymous
4873           \def\bbl@tempc{{##3}{##4}}%
4874         \fi
4875         \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
4876       \fi}%
4877     \bbl@languages
4878     \@ifundefined{bbl@hyphendata@\the\language}%
4879       {\bbl@info{No hyphenation patterns were set for\\%
4880                  language '\bbl@tempa'. Reported}}%
4881       {\expandafter\expandafter\expandafter\bbl@luapatterns
```

171

```
4882            \csname bbl@hyphendata@\the\language\endcsname}}{}}
4883 \endinput\fi
4884   % Here ends \ifx\AddBabelHook\@undefined
4885   % A few lines are only read by hyphen.cfg
4886 \ifx\DisableBabelHook\@undefined
4887   \AddBabelHook{luatex}{everylanguage}{%
4888     \def\process@language##1##2##3{%
4889       \def\process@line####1####2 ####3 ####4 {}}}
4890   \AddBabelHook{luatex}{loadpatterns}{%
4891     \input #1\relax
4892     \expandafter\gdef\csname bbl@hyphendata@\the\language\endcsname
4893       {{#1}{}}}
4894   \AddBabelHook{luatex}{loadexceptions}{%
4895     \input #1\relax
4896     \def\bbl@tempb##1##2{{##1}{#1}}%
4897     \expandafter\xdef\csname bbl@hyphendata@\the\language\endcsname
4898       {\expandafter\expandafter\expandafter\bbl@tempb
4899        \csname bbl@hyphendata@\the\language\endcsname}}
4900 \endinput\fi
4901   % Here stops reading code for hyphen.cfg
4902   % The following is read the 2nd time it's loaded
4903 \begingroup  % TODO - to a lua file
4904 \catcode`\%=12
4905 \catcode`\'=12
4906 \catcode`\"=12
4907 \catcode`\:=12
4908 \directlua{
4909   Babel = Babel or {}
4910   function Babel.bytes(line)
4911     return line:gsub("(.)",
4912       function (chr) return unicode.utf8.char(string.byte(chr)) end)
4913   end
4914   function Babel.begin_process_input()
4915     if luatexbase and luatexbase.add_to_callback then
4916       luatexbase.add_to_callback('process_input_buffer',
4917                                   Babel.bytes,'Babel.bytes')
4918     else
4919       Babel.callback = callback.find('process_input_buffer')
4920       callback.register('process_input_buffer',Babel.bytes)
4921     end
4922   end
4923   function Babel.end_process_input ()
4924     if luatexbase and luatexbase.remove_from_callback then
4925       luatexbase.remove_from_callback('process_input_buffer','Babel.bytes')
4926     else
4927       callback.register('process_input_buffer',Babel.callback)
4928     end
4929   end
4930   function Babel.addpatterns(pp, lg)
4931     local lg = lang.new(lg)
4932     local pats = lang.patterns(lg) or ''
4933     lang.clear_patterns(lg)
4934     for p in pp:gmatch('[^%s]+') do
4935       ss = ''
4936       for i in string.utfcharacters(p:gsub('%d', '')) do
4937         ss = ss .. '%d?' .. i
4938       end
4939       ss = ss:gsub('^%%d%?%.', '%%.') .. '%d?'
4940       ss = ss:gsub('%.%%d%?$', '%%.')
```

```
4941        pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
4942        if n == 0 then
4943          tex.sprint(
4944            [[\string\csname\space bbl@info\endcsname{New pattern: ]]
4945            .. p .. [[}]])
4946          pats = pats .. ' ' .. p
4947        else
4948          tex.sprint(
4949            [[\string\csname\space bbl@info\endcsname{Renew pattern: ]]
4950            .. p .. [[}]])
4951        end
4952      end
4953      lang.patterns(lg, pats)
4954  end
4955 }
4956 \endgroup
4957 \ifx\newattribute\@undefined\else
4958   \newattribute\bbl@attr@locale
4959   \directlua{ Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale' }
4960   \AddBabelHook{luatex}{beforeextras}{%
4961     \setattribute\bbl@attr@locale\localeid}
4962 \fi
4963 \def\BabelStringsDefault{unicode}
4964 \let\luabbl@stop\relax
4965 \AddBabelHook{luatex}{encodedcommands}{%
4966   \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
4967   \ifx\bbl@tempa\bbl@tempb\else
4968     \directlua{Babel.begin_process_input()}%
4969     \def\luabbl@stop{%
4970       \directlua{Babel.end_process_input()}}%
4971   \fi}%
4972 \AddBabelHook{luatex}{stopcommands}{%
4973   \luabbl@stop
4974   \let\luabbl@stop\relax}
4975 \AddBabelHook{luatex}{patterns}{%
4976   \@ifundefined{bbl@hyphendata@\the\language}%
4977     {\def\bbl@elt##1##2##3##4{%
4978        \ifnum##2=\csname l@##2\endcsname % #2=spanish, dutch:OT1...
4979          \def\bbl@tempb{##3}%
4980          \ifx\bbl@tempb\@empty\else % if not a synonymous
4981            \def\bbl@tempc{{##3}{##4}}%
4982          \fi
4983          \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
4984        \fi}%
4985      \bbl@languages
4986      \@ifundefined{bbl@hyphendata@\the\language}%
4987        {\bbl@info{No hyphenation patterns were set for\\%
4988                   language '#2'. Reported}}%
4989        {\expandafter\expandafter\expandafter\bbl@luapatterns
4990          \csname bbl@hyphendata@\the\language\endcsname}}{}%
4991   \@ifundefined{bbl@patterns@}{}{%
4992     \begingroup
4993       \bbl@xin@{,\number\language,}{,\bbl@pttnlist}%
4994       \ifin@\else
4995         \ifx\bbl@patterns@\@empty\else
4996           \directlua{ Babel.addpatterns(
4997             [[\bbl@patterns@]], \number\language) }%
4998         \fi
4999         \@ifundefined{bbl@patterns@#1}%
```

173

```
5000            \@empty
5001            {\directlua{ Babel.addpatterns(
5002                [[\space\csname bbl@patterns@#1\endcsname]],
5003                \number\language) }}%
5004          \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
5005        \fi
5006      \endgroup}%
5007  \bbl@exp{%
5008    \bbl@ifunset{bbl@prehc@\languagename}{}%
5009      {\\\bbl@ifblank{\bbl@cs{prehc@\languagename}}{}%
5010        {\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.

```
5011 \@onlypreamble\babelpatterns
5012 \AtEndOfPackage{%
5013   \newcommand\babelpatterns[2][\@empty]{%
5014     \ifx\bbl@patterns@\relax
5015       \let\bbl@patterns@\@empty
5016     \fi
5017     \ifx\bbl@pttnlist\@empty\else
5018       \bbl@warning{%
5019         You must not intermingle \string\selectlanguage\space and\\%
5020         \string\babelpatterns\space or some patterns will not\\%
5021         be taken into account. Reported}%
5022     \fi
5023     \ifx\@empty#1%
5024       \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
5025     \else
5026       \edef\bbl@tempb{\zap@space#1 \@empty}%
5027       \bbl@for\bbl@tempa\bbl@tempb{%
5028         \bbl@fixname\bbl@tempa
5029         \bbl@iflanguage\bbl@tempa{%
5030           \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
5031             \@ifundefined{bbl@patterns@\bbl@tempa}%
5032               \@empty
5033               {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
5034           #2}}}%
5035     \fi}}
```

13.4   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.

```
5036 % TODO - to a lua file
5037 \directlua{
5038   Babel = Babel or {}
5039   Babel.linebreaking = Babel.linebreaking or {}
5040   Babel.linebreaking.before = {}
5041   Babel.linebreaking.after = {}
5042   Babel.locale = {} % Free to use, indexed by \localeid
5043   function Babel.linebreaking.add_before(func)
5044     tex.print([[\noexpand\csname bbl@luahyphenate\endcsname]])
5045     table.insert(Babel.linebreaking.before, func)
5046   end
```

```
5047    function Babel.linebreaking.add_after(func)
5048      tex.print([[\noexpand\csname bbl@luahyphenate\endcsname]])
5049      table.insert(Babel.linebreaking.after, func)
5050    end
5051 }
5052 \def\bbl@intraspace#1 #2 #3\@@{%
5053    \directlua{
5054      Babel = Babel or {}
5055      Babel.intraspaces = Babel.intraspaces or {}
5056      Babel.intraspaces['\csname bbl@sbcp@\languagename\endcsname'] = %
5057        {b = #1, p = #2, m = #3}
5058      Babel.locale_props[\the\localeid].intraspace = %
5059        {b = #1, p = #2, m = #3}
5060    }}
5061 \def\bbl@intrapenalty#1\@@{%
5062    \directlua{
5063      Babel = Babel or {}
5064      Babel.intrapenalties = Babel.intrapenalties or {}
5065      Babel.intrapenalties['\csname bbl@sbcp@\languagename\endcsname'] = #1
5066      Babel.locale_props[\the\localeid].intrapenalty = #1
5067    }}
5068 \begingroup
5069 \catcode`\%=12
5070 \catcode`\^=14
5071 \catcode`\'=12
5072 \catcode`\~=12
5073 \gdef\bbl@seaintraspace{^
5074    \let\bbl@seaintraspace\relax
5075    \directlua{
5076      Babel = Babel or {}
5077      Babel.sea_enabled = true
5078      Babel.sea_ranges = Babel.sea_ranges or {}
5079      function Babel.set_chranges (script, chrng)
5080        local c = 0
5081        for s, e in string.gmatch(chrng..' ', '(.-)%.%.(.-)%s') do
5082          Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
5083          c = c + 1
5084        end
5085      end
5086      function Babel.sea_disc_to_space (head)
5087        local sea_ranges = Babel.sea_ranges
5088        local last_char = nil
5089        local quad = 655360      ^% 10 pt = 655360 = 10 * 65536
5090        for item in node.traverse(head) do
5091          local i = item.id
5092          if i == node.id'glyph' then
5093            last_char = item
5094          elseif i == 7 and item.subtype == 3 and last_char
5095              and last_char.char > 0x0C99 then
5096            quad = font.getfont(last_char.font).size
5097            for lg, rg in pairs(sea_ranges) do
5098              if last_char.char > rg[1] and last_char.char < rg[2] then
5099                lg = lg:sub(1, 4)  ^% Remove trailing number of, eg, Cyrl1
5100                local intraspace = Babel.intraspaces[lg]
5101                local intrapenalty = Babel.intrapenalties[lg]
5102                local n
5103                if intrapenalty ~= 0 then
5104                  n = node.new(14, 0)     ^% penalty
5105                  n.penalty = intrapenalty
```

```
5106              node.insert_before(head, item, n)
5107            end
5108            n = node.new(12, 13)        ^% (glue, spaceskip)
5109            node.setglue(n, intraspace.b * quad,
5110                            intraspace.p * quad,
5111                            intraspace.m * quad)
5112            node.insert_before(head, item, n)
5113            node.remove(head, item)
5114          end
5115        end
5116      end
5117    end
5118  end
5119  }^^
5120  \bbl@luahyphenate}
```

13.5 CJK line breaking

Minimal line breaking for CJK scripts, mainly intended for simple documents and short texts as a
secondary language. Only line breaking, with a little stretching for justification, without any attempt
to adjust the spacing. It is based on (but does not strictly follow) the Unicode algorithm.
We first need a little table with the corresponding line breaking properties. A few characters have an
additional key for the width (fullwidth *vs.* halfwidth), not yet used. There is a separate file, defined
below.

```
5121 \catcode`\%=14
5122 \gdef\bbl@cjkintraspace{%
5123  \let\bbl@cjkintraspace\relax
5124  \directlua{
5125    Babel = Babel or {}
5126    require('babel-data-cjk.lua')
5127    Babel.cjk_enabled = true
5128    function Babel.cjk_linebreak(head)
5129      local GLYPH = node.id'glyph'
5130      local last_char = nil
5131      local quad = 655360       % 10 pt = 655360 = 10 * 65536
5132      local last_class = nil
5133      local last_lang = nil
5134
5135      for item in node.traverse(head) do
5136        if item.id == GLYPH then
5137
5138          local lang = item.lang
5139
5140          local LOCALE = node.get_attribute(item,
5141              Babel.attr_locale)
5142          local props = Babel.locale_props[LOCALE]
5143
5144          local class = Babel.cjk_class[item.char].c
5145
5146          if props.cjk_quotes and props.cjk_quotes[item.char] then
5147            class = props.cjk_quotes[item.char]
5148          end
5149
5150          if class == 'cp' then class = 'cl' end % )] as CL
5151          if class == 'id' then class = 'I' end
5152
5153          local br = 0
5154          if class and last_class and Babel.cjk_breaks[last_class][class] then
```

```
5155            br = Babel.cjk_breaks[last_class][class]
5156          end
5157
5158        if br == 1 and props.linebreak == 'c' and
5159            lang ~= \the\l@nohyphenation\space and
5160            last_lang ~= \the\l@nohyphenation then
5161          local intrapenalty = props.intrapenalty
5162          if intrapenalty ~= 0 then
5163            local n = node.new(14, 0)      % penalty
5164            n.penalty = intrapenalty
5165            node.insert_before(head, item, n)
5166          end
5167          local intraspace = props.intraspace
5168          local n = node.new(12, 13)      % (glue, spaceskip)
5169          node.setglue(n, intraspace.b * quad,
5170                          intraspace.p * quad,
5171                          intraspace.m * quad)
5172          node.insert_before(head, item, n)
5173        end
5174
5175        if font.getfont(item.font) then
5176          quad = font.getfont(item.font).size
5177        end
5178        last_class = class
5179        last_lang = lang
5180      else % if penalty, glue or anything else
5181        last_class = nil
5182      end
5183    end
5184    lang.hyphenate(head)
5185  end
5186 }%
5187 \bbl@luahyphenate}
5188 \gdef\bbl@luahyphenate{%
5189 \let\bbl@luahyphenate\relax
5190 \directlua{
5191   luatexbase.add_to_callback('hyphenate',
5192   function (head, tail)
5193     if Babel.linebreaking.before then
5194       for k, func in ipairs(Babel.linebreaking.before)  do
5195         func(head)
5196       end
5197     end
5198     if Babel.cjk_enabled then
5199       Babel.cjk_linebreak(head)
5200     end
5201     lang.hyphenate(head)
5202     if Babel.linebreaking.after then
5203       for k, func in ipairs(Babel.linebreaking.after)  do
5204         func(head)
5205       end
5206     end
5207     if Babel.sea_enabled then
5208       Babel.sea_disc_to_space(head)
5209     end
5210   end,
5211   'Babel.hyphenate')
5212 }
5213 }
```

```
5214 \endgroup
5215 \def\bbl@provide@intraspace{%
5216   \bbl@ifunset{bbl@intsp@\languagename}{}%
5217     {\expandafter\ifx\csname bbl@intsp@\languagename\endcsname\@empty\else
5218       \bbl@xin@{/c}{/\bbl@cl{lnbrk}}%
5219       \ifin@              % cjk
5220         \bbl@cjkintraspace
5221         \directlua{
5222             Babel = Babel or {}
5223             Babel.locale_props = Babel.locale_props or {}
5224             Babel.locale_props[\the\localeid].linebreak = 'c'
5225         }%
5226         \bbl@exp{\\\bbl@intraspace\bbl@cl{intsp}\\\@@}%
5227         \ifx\bbl@KVP@intrapenalty\@nil
5228           \bbl@intrapenalty0\@@
5229         \fi
5230       \else             % sea
5231         \bbl@seaintraspace
5232         \bbl@exp{\\\bbl@intraspace\bbl@cl{intsp}\\\@@}%
5233         \directlua{
5234             Babel = Babel or {}
5235             Babel.sea_ranges = Babel.sea_ranges or {}
5236             Babel.set_chranges('\bbl@cl{sbcp}',
5237                                '\bbl@cl{chrng}')
5238         }%
5239         \ifx\bbl@KVP@intrapenalty\@nil
5240           \bbl@intrapenalty0\@@
5241         \fi
5242       \fi
5243     \fi
5244     \ifx\bbl@KVP@intrapenalty\@nil\else
5245       \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
5246     \fi}}
```

13.6   Arabic justification

```
5247 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5248 \def\bblar@chars{%
5249   0628,0629,062A,062B,062C,062D,062E,062F,0630,0631,0632,0633,%
5250   0634,0635,0636,0637,0638,0639,063A,063B,063C,063D,063E,063F,%
5251   0640,0641,0642,0643,0644,0645,0646,0647,0649}
5252 \def\bblar@elongated{%
5253   0626,0628,062A,062B,0633,0634,0635,0636,063B,%
5254   063C,063D,063E,063F,0641,0642,0643,0644,0646,%
5255   0649,064A}
5256 \begingroup
5257   \catcode`\_=11 \catcode`:=11
5258   \gdef\bblar@nofswarn{\gdef\msg_warning:nnx##1##2##3{}}
5259 \endgroup
5260 \gdef\bbl@arabicjust{%
5261   \let\bbl@arabicjust\relax
5262   \newattribute\bblar@kashida
5263   \directlua{ Babel.attr_kashida = luatexbase.registernumber'bblar@kashida' }%
5264   \bblar@kashida=\z@
5265   \bbl@patchfont{{\bbl@parsejalt}}%
5266   \directlua{
5267     Babel.arabic.elong_map   = Babel.arabic.elong_map or {}
5268     Babel.arabic.elong_map[\the\localeid]   = {}
5269     luatexbase.add_to_callback('post_linebreak_filter',
```

```
5270       Babel.arabic.justify, 'Babel.arabic.justify')
5271     luatexbase.add_to_callback('hpack_filter',
5272       Babel.arabic.justify_hbox, 'Babel.arabic.justify_hbox')
5273   }}%
5274 % Save both node lists to make replacement. TODO. Save also widths to
5275 % make computations
5276 \def\bblar@fetchjalt#1#2#3#4{%
5277   \bbl@exp{\\\bbl@foreach{#1}}{%
5278     \bbl@ifunset{bblar@JE@##1}%
5279       {\setbox\z@\hbox{^^^^200d\char"##1#2}}%
5280       {\setbox\z@\hbox{^^^^200d\char"\@nameuse{bblar@JE@##1}#2}}%
5281     \directlua{%
5282       local last = nil
5283       for item in node.traverse(tex.box[0].head) do
5284         if item.id == node.id'glyph' and item.char > 0x600 and
5285             not (item.char == 0x200D) then
5286           last = item
5287         end
5288       end
5289       Babel.arabic.#3['##1#4'] = last.char
5290     }}}
5291 % Brute force. No rules at all, yet. The ideal: look at jalt table. And
5292 % perhaps other tables (falt?, cswh?). What about kaf? And diacritic
5293 % positioning?
5294 \gdef\bbl@parsejalt{%
5295   \ifx\addfontfeature\@undefined\else
5296     \bbl@xin@{/e}{/\bbl@cl{lnbrk}}%
5297     \ifin@
5298       \directlua{%
5299         if Babel.arabic.elong_map[\the\localeid][\fontid\font] == nil then
5300           Babel.arabic.elong_map[\the\localeid][\fontid\font] = {}
5301           tex.print([[\string\csname\space bbl@parsejalti\endcsname]])
5302         end
5303       }%
5304     \fi
5305   \fi}
5306 \gdef\bbl@parsejalti{%
5307   \begingroup
5308     \let\bbl@parsejalt\relax     % To avoid infinite loop
5309     \edef\bbl@tempb{\fontid\font}%
5310     \bblar@nofswarn
5311     \bblar@fetchjalt\bblar@elongated{}{from}{}%
5312     \bblar@fetchjalt\bblar@chars{^^^^064a}{from}{a}% Alef maksura
5313     \bblar@fetchjalt\bblar@chars{^^^^0649}{from}{y}% Yeh
5314     \addfontfeature{RawFeature=+jalt}%
5315     % \@namedef{bblar@JE@0643}{06AA}% todo: catch medial kaf
5316     \bblar@fetchjalt\bblar@elongated{}{dest}{}%
5317     \bblar@fetchjalt\bblar@chars{^^^^064a}{dest}{a}%
5318     \bblar@fetchjalt\bblar@chars{^^^^0649}{dest}{y}%
5319     \directlua{%
5320       for k, v in pairs(Babel.arabic.from) do
5321         if Babel.arabic.dest[k] and
5322             not (Babel.arabic.from[k] == Babel.arabic.dest[k]) then
5323           Babel.arabic.elong_map[\the\localeid][\bbl@tempb]
5324             [Babel.arabic.from[k]] = Babel.arabic.dest[k]
5325         end
5326       end
5327     }%
5328   \endgroup}
```

```
5329 %
5330 \begingroup
5331 \catcode`#=11
5332 \catcode`~=11
5333 \directlua{
5334
5335 Babel.arabic = Babel.arabic or {}
5336 Babel.arabic.from = {}
5337 Babel.arabic.dest = {}
5338 Babel.arabic.justify_factor = 0.95
5339 Babel.arabic.justify_enabled = true
5340
5341 function Babel.arabic.justify(head)
5342   if not Babel.arabic.justify_enabled then return head end
5343   for line in node.traverse_id(node.id'hlist', head) do
5344     Babel.arabic.justify_hlist(head, line)
5345   end
5346   return head
5347 end
5348
5349 function Babel.arabic.justify_hbox(head, gc, size, pack)
5350   local has_inf = false
5351   if Babel.arabic.justify_enabled and pack == 'exactly' then
5352     for n in node.traverse_id(12, head) do
5353       if n.stretch_order > 0 then has_inf = true end
5354     end
5355     if not has_inf then
5356       Babel.arabic.justify_hlist(head, nil, gc, size, pack)
5357     end
5358   end
5359   return head
5360 end
5361
5362 function Babel.arabic.justify_hlist(head, line, gc, size, pack)
5363   local d, new
5364   local k_list, k_item, pos_inline
5365   local width, width_new, full, k_curr, wt_pos, goal, shift
5366   local subst_done = false
5367   local elong_map = Babel.arabic.elong_map
5368   local last_line
5369   local GLYPH = node.id'glyph'
5370   local KASHIDA = Babel.attr_kashida
5371   local LOCALE = Babel.attr_locale
5372
5373   if line == nil then
5374     line = {}
5375     line.glue_sign = 1
5376     line.glue_order = 0
5377     line.head = head
5378     line.shift = 0
5379     line.width = size
5380   end
5381
5382   % Exclude last line. todo. But-- it discards one-word lines, too!
5383   % ? Look for glue = 12:15
5384   if (line.glue_sign == 1 and line.glue_order == 0) then
5385     elongs = {}     % Stores elongated candidates of each line
5386     k_list = {}     % And all letters with kashida
5387     pos_inline = 0  % Not yet used
```

```
5388
5389    for n in node.traverse_id(GLYPH, line.head) do
5390      pos_inline = pos_inline + 1 % To find where it is. Not used.
5391
5392      % Elongated glyphs
5393      if elong_map then
5394        local locale = node.get_attribute(n, LOCALE)
5395        if elong_map[locale] and elong_map[locale][n.font] and
5396            elong_map[locale][n.font][n.char] then
5397          table.insert(elongs, {node = n, locale = locale} )
5398          node.set_attribute(n.prev, KASHIDA, 0)
5399        end
5400      end
5401
5402      % Tatwil
5403      if Babel.kashida_wts then
5404        local k_wt = node.get_attribute(n, KASHIDA)
5405        if k_wt > 0 then % todo. parameter for multi inserts
5406          table.insert(k_list, {node = n, weight = k_wt, pos = pos_inline})
5407        end
5408      end
5409
5410    end % of node.traverse_id
5411
5412    if #elongs == 0 and #k_list == 0 then goto next_line end
5413    full  = line.width
5414    shift = line.shift
5415    goal  = full * Babel.arabic.justify_factor % A bit crude
5416    width = node.dimensions(line.head)    % The 'natural' width
5417
5418    % == Elongated ==
5419    % Original idea taken from 'chikenize'
5420    while (#elongs > 0 and width < goal) do
5421      subst_done = true
5422      local x = #elongs
5423      local curr = elongs[x].node
5424      local oldchar = curr.char
5425      curr.char = elong_map[elongs[x].locale][curr.font][curr.char]
5426      width = node.dimensions(line.head)  % Check if the line is too wide
5427      % Substitute back if the line would be too wide and break:
5428      if width > goal then
5429        curr.char = oldchar
5430        break
5431      end
5432      % If continue, pop the just substituted node from the list:
5433      table.remove(elongs, x)
5434    end
5435
5436    % == Tatwil ==
5437    if #k_list == 0 then goto next_line end
5438
5439    width = node.dimensions(line.head)    % The 'natural' width
5440    k_curr = #k_list
5441    wt_pos = 1
5442
5443    while width < goal do
5444      subst_done = true
5445      k_item = k_list[k_curr].node
5446      if k_list[k_curr].weight == Babel.kashida_wts[wt_pos] then
```

```
5447        d = node.copy(k_item)
5448        d.char = 0x0640
5449        line.head, new = node.insert_after(line.head, k_item, d)
5450        width_new = node.dimensions(line.head)
5451        if width > goal or width == width_new then
5452          node.remove(line.head, new) % Better compute before
5453          break
5454        end
5455        width = width_new
5456      end
5457      if k_curr == 1 then
5458        k_curr = #k_list
5459        wt_pos = (wt_pos >= table.getn(Babel.kashida_wts)) and 1 or wt_pos+1
5460      else
5461        k_curr = k_curr - 1
5462      end
5463    end
5464
5465    ::next_line::
5466
5467    % Must take into account marks and ins, see luatex manual.
5468    % Have to be executed only if there are changes. Investigate
5469    % what's going on exactly.
5470    if subst_done and not gc then
5471      d = node.hpack(line.head, full, 'exactly')
5472      d.shift = shift
5473      node.insert_before(head, line, d)
5474      node.remove(head, line)
5475    end
5476  end % if process line
5477 end
5478 }
5479 \endgroup
5480 \fi\fi % Arabic just block
```

13.7 Common stuff

```
5481 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
5482 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
5483 \DisableBabelHook{babel-fontspec}
5484 ⟨⟨Font selection⟩⟩
```

13.8 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.

```
5485 % TODO - to a lua file
5486 \directlua{
5487 Babel.script_blocks = {
5488   ['dflt'] = {},
5489   ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
5490              {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
5491   ['Armn'] = {{0x0530, 0x058F}},
5492   ['Beng'] = {{0x0980, 0x09FF}},
5493   ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
```

```
5494   ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
5495   ['Cyrl'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
5496              {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
5497   ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
5498   ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
5499              {0xAB00, 0xAB2F}},
5500   ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
5501   % Don't follow strictly Unicode, which places some Coptic letters in
5502   % the 'Greek and Coptic' block
5503   ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
5504   ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
5505              {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
5506              {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
5507              {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
5508              {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
5509              {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
5510   ['Hebr'] = {{0x0590, 0x05FF}},
5511   ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
5512              {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
5513   ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
5514   ['Knda'] = {{0x0C80, 0x0CFF}},
5515   ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
5516              {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
5517              {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
5518   ['Laoo'] = {{0x0E80, 0x0EFF}},
5519   ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
5520              {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
5521              {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
5522   ['Mahj'] = {{0x11150, 0x1117F}},
5523   ['Mlym'] = {{0x0D00, 0x0D7F}},
5524   ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
5525   ['Orya'] = {{0x0B00, 0x0B7F}},
5526   ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
5527   ['Syrc'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
5528   ['Taml'] = {{0x0B80, 0x0BFF}},
5529   ['Telu'] = {{0x0C00, 0x0C7F}},
5530   ['Tfng'] = {{0x2D30, 0x2D7F}},
5531   ['Thai'] = {{0x0E00, 0x0E7F}},
5532   ['Tibt'] = {{0x0F00, 0x0FFF}},
5533   ['Vaii'] = {{0xA500, 0xA63F}},
5534   ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
5535 }
5536
5537 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
5538 Babel.script_blocks.Hant = Babel.script_blocks.Hans
5539 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
5540
5541 function Babel.locale_map(head)
5542   if not Babel.locale_mapped then return head end
5543
5544   local LOCALE = Babel.attr_locale
5545   local GLYPH = node.id('glyph')
5546   local inmath = false
5547   local toloc_save
5548   for item in node.traverse(head) do
5549     local toloc
5550     if not inmath and item.id == GLYPH then
5551       % Optimization: build a table with the chars found
5552       if Babel.chr_to_loc[item.char] then
```

183

```
5553        toloc = Babel.chr_to_loc[item.char]
5554      else
5555        for lc, maps in pairs(Babel.loc_to_scr) do
5556          for _, rg in pairs(maps) do
5557            if item.char >= rg[1] and item.char <= rg[2] then
5558              Babel.chr_to_loc[item.char] = lc
5559              toloc = lc
5560              break
5561            end
5562          end
5563        end
5564      end
5565      % Now, take action, but treat composite chars in a different
5566      % fashion, because they 'inherit' the previous locale. Not yet
5567      % optimized.
5568      if not toloc and
5569          (item.char >= 0x0300 and item.char <= 0x036F) or
5570          (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
5571          (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
5572        toloc = toloc_save
5573      end
5574      if toloc and toloc > -1 then
5575        if Babel.locale_props[toloc].lg then
5576          item.lang = Babel.locale_props[toloc].lg
5577          node.set_attribute(item, LOCALE, toloc)
5578        end
5579        if Babel.locale_props[toloc]['/'..item.font] then
5580          item.font = Babel.locale_props[toloc]['/'..item.font]
5581        end
5582        toloc_save = toloc
5583      end
5584    elseif not inmath and item.id == 7 then
5585      item.replace = item.replace and Babel.locale_map(item.replace)
5586      item.pre     = item.pre and Babel.locale_map(item.pre)
5587      item.post    = item.post and Babel.locale_map(item.post)
5588    elseif item.id == node.id'math' then
5589      inmath = (item.subtype == 0)
5590    end
5591  end
5592  return head
5593 end
5594 }
```

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

```
5595 \newcommand\babelcharproperty[1]{%
5596   \count@=#1\relax
5597   \ifvmode
5598     \expandafter\bbl@chprop
5599   \else
5600     \bbl@error{\string\babelcharproperty\space can be used only in\\%
5601                vertical mode (preamble or between paragraphs)}%
5602                {See the manual for futher info}%
5603   \fi}
5604 \newcommand\bbl@chprop[3][\the\count@]{%
5605   \@tempcnta=#1\relax
5606   \bbl@ifunset{bbl@chprop@#2}%
5607     {\bbl@error{No property named '#2'. Allowed values are\\%
5608                direction (bc), mirror (bmg), and linebreak (lb)}%
```

```
5609                    {See the manual for futher info}}%
5610      {}%
5611    \loop
5612      \bbl@cs{chprop@#2}{#3}%
5613    \ifnum\count@<\@tempcnta
5614      \advance\count@\@ne
5615    \repeat}
5616 \def\bbl@chprop@direction#1{%
5617    \directlua{
5618      Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5619      Babel.characters[\the\count@]['d'] = '#1'
5620    }}
5621 \let\bbl@chprop@bc\bbl@chprop@direction
5622 \def\bbl@chprop@mirror#1{%
5623    \directlua{
5624      Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5625      Babel.characters[\the\count@]['m'] = '\number#1'
5626    }}
5627 \let\bbl@chprop@bmg\bbl@chprop@mirror
5628 \def\bbl@chprop@linebreak#1{%
5629    \directlua{
5630      Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
5631      Babel.cjk_characters[\the\count@]['c'] = '#1'
5632    }}
5633 \let\bbl@chprop@lb\bbl@chprop@linebreak
5634 \def\bbl@chprop@locale#1{%
5635    \directlua{
5636      Babel.chr_to_loc = Babel.chr_to_loc or {}
5637      Babel.chr_to_loc[\the\count@] =
5638        \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@@#1}}\space
5639    }}
```

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.

```
5640 \directlua{
5641    Babel.nohyphenation = \the\l@nohyphenation
5642 }
```

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).

```
5643 \begingroup
5644 \catcode`\~=12
5645 \catcode`\%=12
5646 \catcode`\&=14
5647 \gdef\babelposthyphenation#1#2#3{&%
5648    \bbl@activateposthyphen
5649    \begingroup
5650      \def\babeltempa{\bbl@add@list\babeltempb}&%
5651      \let\babeltempb\@empty
5652      \def\bbl@tempa{#3}&% TODO. Ugly trick to preserve {}:
5653      \bbl@replace\bbl@tempa{,}{ ,}&%
5654      \expandafter\bbl@foreach\expandafter{\bbl@tempa}{&%
```

```
5655        \bbl@ifsamestring{##1}{remove}&%
5656          {\bbl@add@list\babeltempb{nil}}&%
5657          {\directlua{
5658            local rep = [=[##1]=]
5659            rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
5660            rep = rep:gsub('^%s*(insert)%s*,', 'insert = true, ')
5661            rep = rep:gsub(    '(no)%s*=%s*([^%s,]*)', Babel.capture_func)
5662            rep = rep:gsub(   '(pre)%s*=%s*([^%s,]*)', Babel.capture_func)
5663            rep = rep:gsub(  '(post)%s*=%s*([^%s,]*)', Babel.capture_func)
5664            rep = rep:gsub('(string)%s*=%s*([^%s,]*)', Babel.capture_func)
5665            tex.print([[\string\babeltempa{{]] .. rep .. [[}}]])
5666          }}}&%
5667      \directlua{
5668        local lbkr = Babel.linebreaking.replacements[1]
5669        local u = unicode.utf8
5670        local id = \the\csname l@#1\endcsname
5671        &% Convert pattern:
5672        local patt = string.gsub([==[#2]==], '%s', '')
5673        if not u.find(patt, '()', nil, true) then
5674          patt = '()' .. patt .. '()'
5675        end
5676        patt = string.gsub(patt, '%(%)%^', '^()')
5677        patt = string.gsub(patt, '%$%(%)', '()$')
5678        patt = u.gsub(patt, '{(.)}',
5679              function (n)
5680                return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
5681              end)
5682        patt = u.gsub(patt, '{(%x%x%x%x+)}',
5683              function (n)
5684                return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%%1')
5685              end)
5686        lbkr[id] = lbkr[id] or {}
5687        table.insert(lbkr[id], { pattern = patt, replace = { \babeltempb } })
5688      }&%
5689    \endgroup}
5690 % TODO. Copypaste pattern.
5691 \gdef\babelprehyphenation#1#2#3{&%
5692    \bbl@activateprehyphen
5693    \begingroup
5694      \def\babeltempa{\bbl@add@list\babeltempb}&%
5695      \let\babeltempb\@empty
5696      \def\bbl@tempa{#3}&% TODO. Ugly trick to preserve {}:
5697      \bbl@replace\bbl@tempa{,}{ ,}&%
5698      \expandafter\bbl@foreach\expandafter{\bbl@tempa}{&%
5699        \bbl@ifsamestring{##1}{remove}&%
5700          {\bbl@add@list\babeltempb{nil}}&%
5701          {\directlua{
5702            local rep = [=[##1]=]
5703            rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
5704            rep = rep:gsub('^%s*(insert)%s*,', 'insert = true, ')
5705            rep = rep:gsub('(string)%s*=%s*([^%s,]*)', Babel.capture_func)
5706            rep = rep:gsub('(space)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
5707              'space = {' .. '%2, %3, %4' .. '}')
5708            rep = rep:gsub('(spacefactor)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
5709              'spacefactor = {' .. '%2, %3, %4' .. '}')
5710            rep = rep:gsub('(kashida)%s*=%s*([^%s,]*)', Babel.capture_kashida)
5711            tex.print([[\string\babeltempa{{]] .. rep .. [[}}]])
5712          }}}&%
5713      \directlua{
```

```
5714      local lbkr = Babel.linebreaking.replacements[0]
5715      local u = unicode.utf8
5716      local id = \the\csname bbl@id@@#1\endcsname
5717      &% Convert pattern:
5718      local patt = string.gsub([==[#2]==], '%s', '')
5719      local patt = string.gsub(patt, '|', ' ')
5720      if not u.find(patt, '()', nil, true) then
5721        patt = '()' .. patt .. '()'
5722      end
5723      &% patt = string.gsub(patt, '%(%)%^', '^()')
5724      &% patt = string.gsub(patt, '([^%%])%$%(%)', '%1()$')
5725      patt = u.gsub(patt, '{(.)}',
5726            function (n)
5727              return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
5728            end)
5729      patt = u.gsub(patt, '{(%x%x%x%x+)}',
5730            function (n)
5731              return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%%1')
5732            end)
5733      lbkr[id] = lbkr[id] or {}
5734      table.insert(lbkr[id], { pattern = patt, replace = { \babeltempb } })
5735    }&%
5736  \endgroup}
5737 \endgroup
5738 \def\bbl@activateposthyphen{%
5739   \let\bbl@activateposthyphen\relax
5740   \directlua{
5741     require('babel-transforms.lua')
5742     Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
5743   }}
5744 \def\bbl@activateprehyphen{%
5745   \let\bbl@activateprehyphen\relax
5746   \directlua{
5747     require('babel-transforms.lua')
5748     Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
5749   }}
```

13.9 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.

```
5750 \def\bbl@activate@preotf{%
5751   \let\bbl@activate@preotf\relax  % only once
5752   \directlua{
5753     Babel = Babel or {}
5754     %
5755     function Babel.pre_otfload_v(head)
5756       if Babel.numbers and Babel.digits_mapped then
5757         head = Babel.numbers(head)
5758       end
5759       if Babel.bidi_enabled then
5760         head = Babel.bidi(head, false, dir)
5761       end
5762       return head
5763     end
5764     %
5765     function Babel.pre_otfload_h(head, gc, sz, pt, dir)
```

```
5766        if Babel.numbers and Babel.digits_mapped then
5767          head = Babel.numbers(head)
5768        end
5769        if Babel.bidi_enabled then
5770          head = Babel.bidi(head, false, dir)
5771        end
5772        return head
5773      end
5774      %
5775      luatexbase.add_to_callback('pre_linebreak_filter',
5776        Babel.pre_otfload_v,
5777        'Babel.pre_otfload_v',
5778        luatexbase.priority_in_callback('pre_linebreak_filter',
5779          'luaotfload.node_processor') or nil)
5780      %
5781      luatexbase.add_to_callback('hpack_filter',
5782        Babel.pre_otfload_h,
5783        'Babel.pre_otfload_h',
5784        luatexbase.priority_in_callback('hpack_filter',
5785          'luaotfload.node_processor') or nil)
5786   }}
```

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=.

```
5787 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5788   \let\bbl@beforeforeign\leavevmode
5789   \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
5790   \RequirePackage{luatexbase}
5791   \bbl@activate@preotf
5792   \directlua{
5793     require('babel-data-bidi.lua')
5794     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
5795       require('babel-bidi-basic.lua')
5796     \or
5797       require('babel-bidi-basic-r.lua')
5798     \fi}
5799   % TODO - to locale_props, not as separate attribute
5800   \newattribute\bbl@attr@dir
5801   \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
5802   % TODO. I don't like it, hackish:
5803   \bbl@exp{\output{\bodydir\pagedir\the\output}}
5804   \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
5805 \fi\fi
5806 \chardef\bbl@thetextdir\z@
5807 \chardef\bbl@thepardir\z@
5808 \def\bbl@getluadir#1{%
5809   \directlua{
5810     if tex.#1dir == 'TLT' then
5811       tex.sprint('0')
5812     elseif tex.#1dir == 'TRT' then
5813       tex.sprint('1')
5814     end}}
5815 \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
5816   \ifcase#3\relax
5817     \ifcase\bbl@getluadir{#1}\relax\else
5818       #2 TLT\relax
5819     \fi
5820   \else
```

```
5821        \ifcase\bbl@getluadir{#1}\relax
5822          #2 TRT\relax
5823        \fi
5824    \fi}
5825 \def\bbl@textdir#1{%
5826    \bbl@setluadir{text}\textdir{#1}%
5827    \chardef\bbl@thetextdir#1\relax
5828    \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*3+#1}}
5829 \def\bbl@pardir#1{%
5830    \bbl@setluadir{par}\pardir{#1}%
5831    \chardef\bbl@thepardir#1\relax}
5832 \def\bbl@bodydir{\bbl@setluadir{body}\bodydir}
5833 \def\bbl@pagedir{\bbl@setluadir{page}\pagedir}
5834 \def\bbl@dirparastext{\pardir\the\textdir\relax}%    %%%%
5835 %
5836 \ifnum\bbl@bidimode>\z@
5837    \def\bbl@mathboxdir{%
5838      \ifcase\bbl@thetextdir\relax
5839        \everyhbox{\bbl@mathboxdir@aux L}%
5840      \else
5841        \everyhbox{\bbl@mathboxdir@aux R}%
5842      \fi}
5843    \def\bbl@mathboxdir@aux#1{%
5844      \@ifnextchar\egroup{}{\textdir T#1T\relax}}
5845    \frozen@everymath\expandafter{%
5846      \expandafter\bbl@mathboxdir\the\frozen@everymath}
5847    \frozen@everydisplay\expandafter{%
5848      \expandafter\bbl@mathboxdir\the\frozen@everydisplay}
5849 \fi
```

13.10   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.

```
5850 \bbl@trace{Redefinitions for bidi layout}
5851 \ifx\@eqnnum\@undefined\else
5852    \ifx\bbl@attr@dir\@undefined\else
5853      \edef\@eqnnum{{%
5854        \unexpanded{\ifcase\bbl@attr@dir\else\bbl@textdir\@ne\fi}%
5855        \unexpanded\expandafter{\@eqnnum}}}
5856    \fi
5857 \fi
5858 \ifx\bbl@opt@layout\@nnil\endinput\fi  % if no layout
5859 \ifnum\bbl@bidimode>\z@
5860    \def\bbl@nextfake#1{%  non-local changes, use always inside a group!
5861      \bbl@exp{%
5862        \mathdir\the\bodydir
5863        #1%              Once entered in math, set boxes to restore values
```

```
5864        \<ifmmode>%
5865          \everyvbox{%
5866            \the\everyvbox
5867            \bodydir\the\bodydir
5868            \mathdir\the\mathdir
5869            \everyhbox{\the\everyhbox}%
5870            \everyvbox{\the\everyvbox}}%
5871          \everyhbox{%
5872            \the\everyhbox
5873            \bodydir\the\bodydir
5874            \mathdir\the\mathdir
5875            \everyhbox{\the\everyhbox}%
5876            \everyvbox{\the\everyvbox}}%
5877        \<fi>}}%
5878    \def\@hangfrom#1{%
5879      \setbox\@tempboxa\hbox{{#1}}%
5880      \hangindent\wd\@tempboxa
5881      \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
5882        \shapemode\@ne
5883      \fi
5884      \noindent\box\@tempboxa}
5885  \fi
5886  \IfBabelLayout{tabular}
5887    {\let\bbl@OL@@tabular\@tabular
5888    \bbl@replace\@tabular{$}{\bbl@nextfake$}%
5889    \let\bbl@NL@@tabular\@tabular
5890    \AtBeginDocument{%
5891      \ifx\bbl@NL@@tabular\@tabular\else
5892        \bbl@replace\@tabular{$}{\bbl@nextfake$}%
5893        \let\bbl@NL@@tabular\@tabular
5894      \fi}}
5895    {}
5896  \IfBabelLayout{lists}
5897    {\let\bbl@OL@list\list
5898    \bbl@sreplace\list{\parshape}{\bbl@listparshape}%
5899    \let\bbl@NL@list\list
5900    \def\bbl@listparshape#1#2#3{%
5901      \parshape #1 #2 #3 %
5902      \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
5903        \shapemode\tw@
5904      \fi}}
5905    {}
5906  \IfBabelLayout{graphics}
5907    {\let\bbl@pictresetdir\relax
5908    \def\bbl@pictsetdir#1{%
5909      \ifcase\bbl@thetextdir
5910        \let\bbl@pictresetdir\relax
5911      \else
5912        \ifcase#1\bodydir TLT  % Remember this sets the inner boxes
5913          \or\textdir TLT
5914          \else\bodydir TLT \textdir TLT
5915        \fi
5916        % \(text|par)dir required in pgf:
5917        \def\bbl@pictresetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}%
5918      \fi}%
5919    \ifx\AddToHook\@undefined\else
5920      \AddToHook{env/picture/begin}{\bbl@pictsetdir\tw@}%
5921      \directlua{
5922        Babel.get_picture_dir = true
```

190

```
5923        Babel.picture_has_bidi = 0
5924        function Babel.picture_dir (head)
5925          if not Babel.get_picture_dir then return head end
5926          for item in node.traverse(head) do
5927            if item.id == node.id'glyph' then
5928              local itemchar = item.char
5929              % TODO. Copypaste pattern from Babel.bidi (-r)
5930              local chardata = Babel.characters[itemchar]
5931              local dir = chardata and chardata.d or nil
5932              if not dir then
5933                for nn, et in ipairs(Babel.ranges) do
5934                  if itemchar < et[1] then
5935                    break
5936                  elseif itemchar <= et[2] then
5937                    dir = et[3]
5938                    break
5939                  end
5940                end
5941              end
5942              if dir and (dir == 'al' or dir == 'r') then
5943                Babel.picture_has_bidi = 1
5944              end
5945            end
5946          end
5947          return head
5948        end
5949        luatexbase.add_to_callback("hpack_filter", Babel.picture_dir,
5950          "Babel.picture_dir")
5951      }%
5952    \AtBeginDocument{%
5953      \long\def\put(#1,#2)#3{%
5954        \@killglue
5955        % Try:
5956        \ifx\bbl@pictresetdir\relax
5957          \def\bbl@tempc{0}%
5958        \else
5959          \directlua{
5960            Babel.get_picture_dir = true
5961            Babel.picture_has_bidi = 0
5962          }%
5963          \setbox\z@\hb@xt@\z@{%
5964            \@defaultunitsset\@tempdimc{#1}\unitlength
5965            \kern\@tempdimc
5966            #3\hss}%
5967          \edef\bbl@tempc{\directlua{tex.print(Babel.picture_has_bidi)}}%
5968        \fi
5969        % Do:
5970        \@defaultunitsset\@tempdimc{#2}\unitlength
5971        \raise\@tempdimc\hb@xt@\z@{%
5972          \@defaultunitsset\@tempdimc{#1}\unitlength
5973          \kern\@tempdimc
5974          {\ifnum\bbl@tempc>\z@\bbl@pictresetdir\fi#3}\hss}%
5975        \ignorespaces}%
5976      \MakeRobust\put}%
5977    \fi
5978    \AtBeginDocument
5979      {\ifx\pgfpicture\@undefined\else % TODO. Allow deactivate?
5980        \ifx\AddToHook\@undefined
5981          \bbl@sreplace\pgfpicture{\pgfpicturetrue}%
```

191

```
5982          {\bbl@pictsetdir\z@\pgfpicturetrue}%
5983        \else
5984          \AddToHook{env/pgfpicture/begin}{\bbl@pictsetdir\@ne}%
5985        \fi
5986        \bbl@add\pgfinterruptpicture{\bbl@pictresetdir}%
5987        \bbl@add\pgfsys@beginpicture{\bbl@pictsetdir\z@}%
5988      \fi
5989      \ifx\tikzpicture\@undefined\else
5990        \ifx\AddToHook\@undefined\else
5991          \AddToHook{env/tikzpicture/begin}{\bbl@pictsetdir\z@}%
5992        \fi
5993        \bbl@add\tikz@atbegin@node{\bbl@pictresetdir}%
5994        \bbl@sreplace\tikz{\begingroup}{\begingroup\bbl@pictsetdir\tw@}%
5995      \fi
5996      \ifx\AddToHook\@undefined\else
5997        \ifx\tcolorbox\@undefined\else
5998          \AddToHook{env/tcolorbox/begin}{\bbl@pictsetdir\@ne}%
5999          \bbl@sreplace\tcb@savebox
6000            {\ignorespaces}{\ignorespaces\bbl@pictresetdir}%
6001        \fi
6002      \fi
6003    }}
6004    {}
```

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`.

```
6005 \IfBabelLayout{counters}%
6006    {\let\bbl@OL@@textsuperscript\@textsuperscript
6007     \bbl@sreplace\@textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
6008     \let\bbl@latinarabic=\@arabic
6009     \let\bbl@OL@@arabic\@arabic
6010     \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
6011     \@ifpackagewith{babel}{bidi=default}%
6012       {\let\bbl@asciiroman=\@roman
6013        \let\bbl@OL@@roman\@roman
6014        \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciiroman#1}}}%
6015        \let\bbl@asciiRoman=\@Roman
6016        \let\bbl@OL@@roman\@Roman
6017        \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}%
6018        \let\bbl@OL@labelenumii\labelenumii
6019        \def\labelenumii{)\theenumii(}%
6020        \let\bbl@OL@p@enumiii\p@enumiii
6021        \def\p@enumiii{\p@enumii)\theenumii(}}{}}{}
6022 ⟨⟨Footnote changes⟩⟩
6023 \IfBabelLayout{footnotes}%
6024    {\let\bbl@OL@footnote\footnote
6025     \BabelFootnote\footnote\languagename{}{}%
6026     \BabelFootnote\localfootnote\languagename{}{}%
6027     \BabelFootnote\mainfootnote{}{}{}}
6028    {}
```

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.

```
6029 \IfBabelLayout{extras}%
6030    {\let\bbl@OL@underline\underline
6031     \bbl@sreplace\underline{$\@@underline}{\bbl@nextfake$\@@underline}%
6032     \let\bbl@OL@LaTeX2e\LaTeX2e
6033     \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
```

```
6034      \if b\expandafter\@car\f@series\@nil\boldmath\fi
6035      \babelsublr{%
6036        \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}}}
6037   {}
6038 ⟨/luatex⟩
```

13.11   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.

```
6039 ⟨*transforms⟩
6040 Babel.linebreaking.replacements = {}
6041 Babel.linebreaking.replacements[0] = {}  -- pre
6042 Babel.linebreaking.replacements[1] = {}  -- post
6043
6044 -- Discretionaries contain strings as nodes
6045 function Babel.str_to_nodes(fn, matches, base)
6046   local n, head, last
6047   if fn == nil then return nil end
6048   for s in string.utfvalues(fn(matches)) do
6049     if base.id == 7 then
6050       base = base.replace
6051     end
6052     n = node.copy(base)
6053     n.char     = s
6054     if not head then
6055       head = n
6056     else
6057       last.next = n
6058     end
6059     last = n
6060   end
6061   return head
6062 end
6063
6064 Babel.fetch_subtext = {}
6065
6066 Babel.ignore_pre_char = function(node)
6067   return (node.lang == Babel.nohyphenation)
6068 end
6069
6070 -- Merging both functions doesn't seen feasible, because there are too
6071 -- many differences.
6072 Babel.fetch_subtext[0] = function(head)
6073   local word_string = ''
6074   local word_nodes = {}
6075   local lang
6076   local item = head
6077   local inmath = false
```

193

```
6078
6079   while item do
6080
6081     if item.id == 11 then
6082       inmath = (item.subtype == 0)
6083     end
6084
6085     if inmath then
6086       -- pass
6087
6088     elseif item.id == 29 then
6089       local locale = node.get_attribute(item, Babel.attr_locale)
6090
6091       if lang == locale or lang == nil then
6092         lang = lang or locale
6093         if Babel.ignore_pre_char(item) then
6094           word_string = word_string .. Babel.us_char
6095         else
6096           word_string = word_string .. unicode.utf8.char(item.char)
6097         end
6098         word_nodes[#word_nodes+1] = item
6099       else
6100         break
6101       end
6102
6103     elseif item.id == 12 and item.subtype == 13 then
6104       word_string = word_string .. ' '
6105       word_nodes[#word_nodes+1] = item
6106
6107     -- Ignore leading unrecognized nodes, too.
6108     elseif word_string ~= '' then
6109       word_string = word_string .. Babel.us_char
6110       word_nodes[#word_nodes+1] = item  -- Will be ignored
6111     end
6112
6113     item = item.next
6114   end
6115
6116   -- Here and above we remove some trailing chars but not the
6117   -- corresponding nodes. But they aren't accessed.
6118   if word_string:sub(-1) == ' ' then
6119     word_string = word_string:sub(1,-2)
6120   end
6121   word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
6122   return word_string, word_nodes, item, lang
6123 end
6124
6125 Babel.fetch_subtext[1] = function(head)
6126   local word_string = ''
6127   local word_nodes = {}
6128   local lang
6129   local item = head
6130   local inmath = false
6131
6132   while item do
6133
6134     if item.id == 11 then
6135       inmath = (item.subtype == 0)
6136     end
```

```
6137
6138      if inmath then
6139        -- pass
6140
6141      elseif item.id == 29 then
6142        if item.lang == lang or lang == nil then
6143          if (item.char ~= 124) and (item.char ~= 61) then -- not =, not |
6144            lang = lang or item.lang
6145            word_string = word_string .. unicode.utf8.char(item.char)
6146            word_nodes[#word_nodes+1] = item
6147          end
6148        else
6149          break
6150        end
6151
6152      elseif item.id == 7 and item.subtype == 2 then
6153        word_string = word_string .. '='
6154        word_nodes[#word_nodes+1] = item
6155
6156      elseif item.id == 7 and item.subtype == 3 then
6157        word_string = word_string .. '|'
6158        word_nodes[#word_nodes+1] = item
6159
6160      -- (1) Go to next word if nothing was found, and (2) implicitly
6161      -- remove leading USs.
6162      elseif word_string == '' then
6163        -- pass
6164
6165      -- This is the responsible for splitting by words.
6166      elseif (item.id == 12 and item.subtype == 13) then
6167        break
6168
6169      else
6170        word_string = word_string .. Babel.us_char
6171        word_nodes[#word_nodes+1] = item  -- Will be ignored
6172      end
6173
6174      item = item.next
6175    end
6176
6177    word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
6178    return word_string, word_nodes, item, lang
6179 end
6180
6181 function Babel.pre_hyphenate_replace(head)
6182    Babel.hyphenate_replace(head, 0)
6183 end
6184
6185 function Babel.post_hyphenate_replace(head)
6186    Babel.hyphenate_replace(head, 1)
6187 end
6188
6189 Babel.us_char = string.char(31)
6190
6191 function Babel.hyphenate_replace(head, mode)
6192    local u = unicode.utf8
6193    local lbkr = Babel.linebreaking.replacements[mode]
6194
6195    local word_head = head
```

```
6196
6197   while true do  -- for each subtext block
6198
6199     local w, w_nodes, nw, lang = Babel.fetch_subtext[mode](word_head)
6200
6201     if Babel.debug then
6202       print()
6203       print((mode == 0) and '@@@@<' or '@@@@>', w)
6204     end
6205
6206     if nw == nil and w == '' then break end
6207
6208     if not lang then goto next end
6209     if not lbkr[lang] then goto next end
6210
6211     -- For each saved (pre|post)hyphenation. TODO. Reconsider how
6212     -- loops are nested.
6213     for k=1, #lbkr[lang] do
6214       local p = lbkr[lang][k].pattern
6215       local r = lbkr[lang][k].replace
6216
6217       if Babel.debug then
6218         print('*****', p, mode)
6219       end
6220
6221       -- This variable is set in some cases below to the first *byte*
6222       -- after the match, either as found by u.match (faster) or the
6223       -- computed position based on sc if w has changed.
6224       local last_match = 0
6225       local step = 0
6226
6227       -- For every match.
6228       while true do
6229         if Babel.debug then
6230           print('=====')
6231         end
6232         local new  -- used when inserting and removing nodes
6233
6234         local matches = { u.match(w, p, last_match) }
6235
6236         if #matches < 2 then break end
6237
6238         -- Get and remove empty captures (with ()'s, which return a
6239         -- number with the position), and keep actual captures
6240         -- (from (...)), if any, in matches.
6241         local first = table.remove(matches, 1)
6242         local last  = table.remove(matches, #matches)
6243         -- Non re-fetched substrings may contain \31, which separates
6244         -- subsubstrings.
6245         if string.find(w:sub(first, last-1), Babel.us_char) then break end
6246
6247         local save_last = last -- with A()BC()D, points to D
6248
6249         -- Fix offsets, from bytes to unicode. Explained above.
6250         first = u.len(w:sub(1, first-1)) + 1
6251         last  = u.len(w:sub(1, last-1)) -- now last points to C
6252
6253         -- This loop stores in a small table the nodes
6254         -- corresponding to the pattern. Used by 'data' to provide a
```

196

```
6255          -- predictable behavior with 'insert' (w_nodes is modified on
6256          -- the fly), and also access to 'remove'd nodes.
6257          local sc = first-1          -- Used below, too
6258          local data_nodes = {}
6259
6260          for q = 1, last-first+1 do
6261            data_nodes[q] = w_nodes[sc+q]
6262          end
6263
6264          -- This loop traverses the matched substring and takes the
6265          -- corresponding action stored in the replacement list.
6266          -- sc = the position in substr nodes / string
6267          -- rc = the replacement table index
6268          local rc = 0
6269
6270          while rc < last-first+1 do -- for each replacement
6271            if Babel.debug then
6272              print('.....', rc + 1)
6273            end
6274            sc = sc + 1
6275            rc = rc + 1
6276
6277            if Babel.debug then
6278              Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6279              local ss = ''
6280              for itt in node.traverse(head) do
6281                if itt.id == 29 then
6282                  ss = ss .. unicode.utf8.char(itt.char)
6283                else
6284                  ss = ss .. '{' .. itt.id .. '}'
6285                end
6286              end
6287              print('*****************', ss)
6288
6289            end
6290
6291            local crep = r[rc]
6292            local item = w_nodes[sc]
6293            local item_base = item
6294            local placeholder = Babel.us_char
6295            local d
6296
6297            if crep and crep.data then
6298              item_base = data_nodes[crep.data]
6299            end
6300
6301            if crep then
6302              step = crep.step or 0
6303            end
6304
6305            if crep and next(crep) == nil then -- = {}
6306              last_match = save_last    -- Optimization
6307              goto next
6308
6309            elseif crep == nil or crep.remove then
6310              node.remove(head, item)
6311              table.remove(w_nodes, sc)
6312              w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
6313              sc = sc - 1  -- Nothing has been inserted.
```

197

```
6314              last_match = utf8.offset(w, sc+1+step)
6315              goto next
6316
6317          elseif crep and crep.kashida then -- Experimental
6318            node.set_attribute(item,
6319                Babel.attr_kashida,
6320                crep.kashida)
6321            last_match = utf8.offset(w, sc+1+step)
6322            goto next
6323
6324          elseif crep and crep.string then
6325            local str = crep.string(matches)
6326            if str == '' then  -- Gather with nil
6327              node.remove(head, item)
6328              table.remove(w_nodes, sc)
6329              w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
6330              sc = sc - 1  -- Nothing has been inserted.
6331            else
6332              local loop_first = true
6333              for s in string.utfvalues(str) do
6334                d = node.copy(item_base)
6335                d.char = s
6336                if loop_first then
6337                  loop_first = false
6338                  head, new = node.insert_before(head, item, d)
6339                  if sc == 1 then
6340                    word_head = head
6341                  end
6342                  w_nodes[sc] = d
6343                  w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc+1)
6344                else
6345                  sc = sc + 1
6346                  head, new = node.insert_before(head, item, d)
6347                  table.insert(w_nodes, sc, new)
6348                  w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc)
6349                end
6350                if Babel.debug then
6351                  print('.....', 'str')
6352                  Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6353                end
6354              end  -- for
6355              node.remove(head, item)
6356            end  -- if ''
6357            last_match = utf8.offset(w, sc+1+step)
6358            goto next
6359
6360          elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
6361            d = node.new(7, 0)   -- (disc, discretionary)
6362            d.pre     = Babel.str_to_nodes(crep.pre, matches, item_base)
6363            d.post    = Babel.str_to_nodes(crep.post, matches, item_base)
6364            d.replace = Babel.str_to_nodes(crep.no, matches, item_base)
6365            d.attr = item_base.attr
6366            if crep.pre == nil then  -- TeXbook p96
6367              d.penalty = crep.penalty or tex.hyphenpenalty
6368            else
6369              d.penalty = crep.penalty or tex.exhyphenpenalty
6370            end
6371            placeholder = '|'
6372            head, new = node.insert_before(head, item, d)
```

```
6373
6374            elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
6375              -- ERROR
6376
6377            elseif crep and crep.penalty then
6378              d = node.new(14, 0)    -- (penalty, userpenalty)
6379              d.attr = item_base.attr
6380              d.penalty = crep.penalty
6381              head, new = node.insert_before(head, item, d)
6382
6383            elseif crep and crep.space then
6384              -- 655360 = 10 pt = 10 * 65536 sp
6385              d = node.new(12, 13)       -- (glue, spaceskip)
6386              local quad = font.getfont(item_base.font).size or 655360
6387              node.setglue(d, crep.space[1] * quad,
6388                              crep.space[2] * quad,
6389                              crep.space[3] * quad)
6390              if mode == 0 then
6391                placeholder = ' '
6392              end
6393              head, new = node.insert_before(head, item, d)
6394
6395            elseif crep and crep.spacefactor then
6396              d = node.new(12, 13)        -- (glue, spaceskip)
6397              local base_font = font.getfont(item_base.font)
6398              node.setglue(d,
6399                crep.spacefactor[1] * base_font.parameters['space'],
6400                crep.spacefactor[2] * base_font.parameters['space_stretch'],
6401                crep.spacefactor[3] * base_font.parameters['space_shrink'])
6402              if mode == 0 then
6403                placeholder = ' '
6404              end
6405              head, new = node.insert_before(head, item, d)
6406
6407            elseif mode == 0 and crep and crep.space then
6408              -- ERROR
6409
6410            end  -- ie replacement cases
6411
6412            -- Shared by disc, space and penalty.
6413            if sc == 1 then
6414              word_head = head
6415            end
6416            if crep.insert then
6417              w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc)
6418              table.insert(w_nodes, sc, new)
6419              last = last + 1
6420            else
6421              w_nodes[sc] = d
6422              node.remove(head, item)
6423              w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1)
6424            end
6425
6426            last_match = utf8.offset(w, sc+1+step)
6427
6428            ::next::
6429
6430        end  -- for each replacement
6431
```

```
6432        if Babel.debug then
6433            print('.....', '/')
6434            Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6435        end
6436
6437      end  -- for match
6438
6439    end  -- for patterns
6440
6441    ::next::
6442    word_head = nw
6443  end  -- for substring
6444  return head
6445 end
6446
6447 -- This table stores capture maps, numbered consecutively
6448 Babel.capture_maps = {}
6449
6450 -- The following functions belong to the next macro
6451 function Babel.capture_func(key, cap)
6452  local ret = "[[" .. cap:gsub('{([0-9])}', "]]..m[%1]..[[") .. "]]"
6453  local cnt
6454  local u = unicode.utf8
6455  ret, cnt = ret:gsub('{([0-9])}|([^|]+)|(.-)}', Babel.capture_func_map)
6456  if cnt == 0 then
6457    ret = u.gsub(ret, '{(%x%x%x%x+)}',
6458          function (n)
6459            return u.char(tonumber(n, 16))
6460          end)
6461  end
6462  ret = ret:gsub("%[%[%]%]%.%.", '')
6463  ret = ret:gsub("%.%.%[%[%]%]", '')
6464  return key .. [[=function(m) return ]] .. ret .. [[ end]]
6465 end
6466
6467 function Babel.capt_map(from, mapno)
6468  return Babel.capture_maps[mapno][from] or from
6469 end
6470
6471 -- Handle the {n|abc|ABC} syntax in captures
6472 function Babel.capture_func_map(capno, from, to)
6473  local u = unicode.utf8
6474  from = u.gsub(from, '{(%x%x%x%x+)}',
6475        function (n)
6476          return u.char(tonumber(n, 16))
6477        end)
6478  to = u.gsub(to, '{(%x%x%x%x+)}',
6479        function (n)
6480          return u.char(tonumber(n, 16))
6481        end)
6482  local froms = {}
6483  for s in string.utfcharacters(from) do
6484    table.insert(froms, s)
6485  end
6486  local cnt = 1
6487  table.insert(Babel.capture_maps, {})
6488  local mlen = table.getn(Babel.capture_maps)
6489  for s in string.utfcharacters(to) do
6490    Babel.capture_maps[mlen][froms[cnt]] = s
```

```
6491    cnt = cnt + 1
6492  end
6493  return "]]..Babel.capt_map(m[" .. capno .. "]," ..
6494        (mlen) .. ")..' .. "[["
6495 end
6496
6497 -- Create/Extend reversed sorted list of kashida weights:
6498 function Babel.capture_kashida(key, wt)
6499  wt = tonumber(wt)
6500  if Babel.kashida_wts then
6501    for p, q in ipairs(Babel.kashida_wts) do
6502      if wt  == q then
6503        break
6504      elseif wt > q then
6505        table.insert(Babel.kashida_wts, p, wt)
6506        break
6507      elseif table.getn(Babel.kashida_wts) == p then
6508        table.insert(Babel.kashida_wts, wt)
6509      end
6510    end
6511  else
6512    Babel.kashida_wts = { wt }
6513  end
6514  return 'kashida = ' .. wt
6515 end
6516 ⟨/transforms⟩
```

13.12   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>).

201

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.

```
6517 ⟨∗basic-r⟩
6518 Babel = Babel or {}
6519
6520 Babel.bidi_enabled = true
6521
6522 require('babel-data-bidi.lua')
6523
6524 local characters = Babel.characters
6525 local ranges = Babel.ranges
6526
6527 local DIR = node.id("dir")
6528
6529 local function dir_mark(head, from, to, outer)
6530   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
6531   local d = node.new(DIR)
6532   d.dir = '+' .. dir
6533   node.insert_before(head, from, d)
6534   d = node.new(DIR)
6535   d.dir = '-' .. dir
6536   node.insert_after(head, to, d)
6537 end
6538
6539 function Babel.bidi(head, ispar)
6540   local first_n, last_n          -- first and last char with nums
6541   local last_es                  -- an auxiliary 'last' used with nums
6542   local first_d, last_d          -- first and last char in L/R block
6543   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):

```
6544   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
6545   local strong_lr = (strong == 'l') and 'l' or 'r'
6546   local outer = strong
6547
6548   local new_dir = false
6549   local first_dir = false
6550   local inmath = false
6551
6552   local last_lr
6553
6554   local type_n = ''
6555
6556   for item in node.traverse(head) do
6557
6558     -- three cases: glyph, dir, otherwise
6559     if item.id == node.id'glyph'
6560       or (item.id == 7 and item.subtype == 2) then
6561
6562       local itemchar
6563       if item.id == 7 and item.subtype == 2 then
6564         itemchar = item.replace.char
```

```
6565        else
6566          itemchar = item.char
6567        end
6568        local chardata = characters[itemchar]
6569        dir = chardata and chardata.d or nil
6570        if not dir then
6571          for nn, et in ipairs(ranges) do
6572            if itemchar < et[1] then
6573              break
6574            elseif itemchar <= et[2] then
6575              dir = et[3]
6576              break
6577            end
6578          end
6579        end
6580        dir = dir or 'l'
6581        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).

```
6582        if new_dir then
6583          attr_dir = 0
6584          for at in node.traverse(item.attr) do
6585            if at.number == Babel.attr_dir then
6586              attr_dir = at.value % 3
6587            end
6588          end
6589          if attr_dir == 1 then
6590            strong = 'r'
6591          elseif attr_dir == 2 then
6592            strong = 'al'
6593          else
6594            strong = 'l'
6595          end
6596          strong_lr = (strong == 'l') and 'l' or 'r'
6597          outer = strong_lr
6598          new_dir = false
6599        end
6600
6601        if dir == 'nsm' then dir = strong end               -- W1
```

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

```
6602        dir_real = dir               -- We need dir_real to set strong below
6603        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:

```
6604        if strong == 'al' then
6605          if dir == 'en' then dir = 'an' end               -- W2
6606          if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
6607          strong_lr = 'r'                                   -- W3
6608        end
```

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

```
6609      elseif item.id == node.id'dir' and not inmath then
6610        new_dir = true
```

```
6611      dir = nil
6612    elseif item.id == node.id'math' then
6613      inmath = (item.subtype == 0)
6614    else
6615      dir = nil            -- Not a char
6616    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>.

```
6617    if dir == 'en' or dir == 'an' or dir == 'et' then
6618      if dir ~= 'et' then
6619        type_n = dir
6620      end
6621      first_n = first_n or item
6622      last_n = last_es or item
6623      last_es = nil
6624    elseif dir == 'es' and last_n then -- W3+W6
6625      last_es = item
6626    elseif dir == 'cs' then              -- it's right - do nothing
6627    elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
6628      if strong_lr == 'r' and type_n ~= '' then
6629        dir_mark(head, first_n, last_n, 'r')
6630      elseif strong_lr == 'l' and first_d and type_n == 'an' then
6631        dir_mark(head, first_n, last_n, 'r')
6632        dir_mark(head, first_d, last_d, outer)
6633        first_d, last_d = nil, nil
6634      elseif strong_lr == 'l' and type_n ~= '' then
6635        last_d = last_n
6636      end
6637      type_n = ''
6638      first_n, last_n = nil, nil
6639    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:

```
6640    if dir == 'l' or dir == 'r' then
6641      if dir ~= outer then
6642        first_d = first_d or item
6643        last_d = item
6644      elseif first_d and dir ~= strong_lr then
6645        dir_mark(head, first_d, last_d, outer)
6646        first_d, last_d = nil, nil
6647      end
6648    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.

```
6649    if dir and not last_lr and dir ~= 'l' and outer == 'r' then
6650      item.char = characters[item.char] and
6651                  characters[item.char].m or item.char
6652    elseif (dir or new_dir) and last_lr ~= item then
```

```
6653        local mir = outer .. strong_lr .. (dir or outer)
6654        if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
6655          for ch in node.traverse(node.next(last_lr)) do
6656            if ch == item then break end
6657            if ch.id == node.id'glyph' and characters[ch.char] then
6658              ch.char = characters[ch.char].m or ch.char
6659            end
6660          end
6661        end
6662      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`).

```
6663      if dir == 'l' or dir == 'r' then
6664        last_lr = item
6665        strong = dir_real          -- Don't search back - best save now
6666        strong_lr = (strong == 'l') and 'l' or 'r'
6667      elseif new_dir then
6668        last_lr = nil
6669      end
6670    end
```

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

```
6671    if last_lr and outer == 'r' then
6672      for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
6673        if characters[ch.char] then
6674          ch.char = characters[ch.char].m or ch.char
6675        end
6676      end
6677    end
6678    if first_n then
6679      dir_mark(head, first_n, last_n, outer)
6680    end
6681    if first_d then
6682      dir_mark(head, first_d, last_d, outer)
6683    end
```

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

```
6684    return node.prev(head) or head
6685 end
6686 ⟨/basic-r⟩
```

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

```
6687 ⟨*basic⟩
6688 Babel = Babel or {}
6689
6690 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
6691
6692 Babel.fontmap = Babel.fontmap or {}
6693 Babel.fontmap[0] = {}       -- l
6694 Babel.fontmap[1] = {}       -- r
6695 Babel.fontmap[2] = {}       -- al/an
6696
6697 Babel.bidi_enabled = true
6698 Babel.mirroring_enabled = true
6699
6700 require('babel-data-bidi.lua')
6701
```

```
6702 local characters = Babel.characters
6703 local ranges = Babel.ranges
6704
6705 local DIR = node.id('dir')
6706 local GLYPH = node.id('glyph')
6707
6708 local function insert_implicit(head, state, outer)
6709   local new_state = state
6710   if state.sim and state.eim and state.sim ~= state.eim then
6711     dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
6712     local d = node.new(DIR)
6713     d.dir = '+' .. dir
6714     node.insert_before(head, state.sim, d)
6715     local d = node.new(DIR)
6716     d.dir = '-' .. dir
6717     node.insert_after(head, state.eim, d)
6718   end
6719   new_state.sim, new_state.eim = nil, nil
6720   return head, new_state
6721 end
6722
6723 local function insert_numeric(head, state)
6724   local new
6725   local new_state = state
6726   if state.san and state.ean and state.san ~= state.ean then
6727     local d = node.new(DIR)
6728     d.dir = '+TLT'
6729     _, new = node.insert_before(head, state.san, d)
6730     if state.san == state.sim then state.sim = new end
6731     local d = node.new(DIR)
6732     d.dir = '-TLT'
6733     _, new = node.insert_after(head, state.ean, d)
6734     if state.ean == state.eim then state.eim = new end
6735   end
6736   new_state.san, new_state.ean = nil, nil
6737   return head, new_state
6738 end
6739
6740 -- TODO - \hbox with an explicit dir can lead to wrong results
6741 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
6742 -- was s made to improve the situation, but the problem is the 3-dir
6743 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
6744 -- well.
6745
6746 function Babel.bidi(head, ispar, hdir)
6747   local d    -- d is used mainly for computations in a loop
6748   local prev_d = ''
6749   local new_d = false
6750
6751   local nodes = {}
6752   local outer_first = nil
6753   local inmath = false
6754
6755   local glue_d = nil
6756   local glue_i = nil
6757
6758   local has_en = false
6759   local first_et = nil
6760
```

```
6761    local ATDIR = Babel.attr_dir
6762
6763    local save_outer
6764    local temp = node.get_attribute(head, ATDIR)
6765    if temp then
6766      temp = temp % 3
6767      save_outer = (temp == 0 and 'l') or
6768                   (temp == 1 and 'r') or
6769                   (temp == 2 and 'al')
6770    elseif ispar then            -- Or error? Shouldn't happen
6771      save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
6772    else                         -- Or error? Shouldn't happen
6773      save_outer = ('TRT' == hdir) and 'r' or 'l'
6774    end
6775      -- when the callback is called, we are just _after_ the box,
6776      -- and the textdir is that of the surrounding text
6777    -- if not ispar and hdir ~= tex.textdir then
6778    --   save_outer = ('TRT' == hdir) and 'r' or 'l'
6779    -- end
6780    local outer = save_outer
6781    local last = outer
6782    -- 'al' is only taken into account in the first, current loop
6783    if save_outer == 'al' then save_outer = 'r' end
6784
6785    local fontmap = Babel.fontmap
6786
6787    for item in node.traverse(head) do
6788
6789      -- In what follows, #node is the last (previous) node, because the
6790      -- current one is not added until we start processing the neutrals.
6791
6792      -- three cases: glyph, dir, otherwise
6793      if item.id == GLYPH
6794         or (item.id == 7 and item.subtype == 2) then
6795
6796        local d_font = nil
6797        local item_r
6798        if item.id == 7 and item.subtype == 2 then
6799          item_r = item.replace    -- automatic discs have just 1 glyph
6800        else
6801          item_r = item
6802        end
6803        local chardata = characters[item_r.char]
6804        d = chardata and chardata.d or nil
6805        if not d or d == 'nsm' then
6806          for nn, et in ipairs(ranges) do
6807            if item_r.char < et[1] then
6808              break
6809            elseif item_r.char <= et[2] then
6810              if not d then d = et[3]
6811              elseif d == 'nsm' then d_font = et[3]
6812              end
6813              break
6814            end
6815          end
6816        end
6817        d = d or 'l'
6818
6819        -- A short 'pause' in bidi for mapfont
```

```
6820        d_font = d_font or d
6821        d_font = (d_font == 'l' and 0) or
6822                 (d_font == 'nsm' and 0) or
6823                 (d_font == 'r' and 1) or
6824                 (d_font == 'al' and 2) or
6825                 (d_font == 'an' and 2) or nil
6826      if d_font and fontmap and fontmap[d_font][item_r.font] then
6827        item_r.font = fontmap[d_font][item_r.font]
6828      end
6829
6830      if new_d then
6831        table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
6832        if inmath then
6833          attr_d = 0
6834        else
6835          attr_d = node.get_attribute(item, ATDIR)
6836          attr_d = attr_d % 3
6837        end
6838        if attr_d == 1 then
6839          outer_first = 'r'
6840          last = 'r'
6841        elseif attr_d == 2 then
6842          outer_first = 'r'
6843          last = 'al'
6844        else
6845          outer_first = 'l'
6846          last = 'l'
6847        end
6848        outer = last
6849        has_en = false
6850        first_et = nil
6851        new_d = false
6852      end
6853
6854      if glue_d then
6855        if (d == 'l' and 'l' or 'r') ~= glue_d then
6856          table.insert(nodes, {glue_i, 'on', nil})
6857        end
6858        glue_d = nil
6859        glue_i = nil
6860      end
6861
6862    elseif item.id == DIR then
6863      d = nil
6864      new_d = true
6865
6866    elseif item.id == node.id'glue' and item.subtype == 13 then
6867      glue_d = d
6868      glue_i = item
6869      d = nil
6870
6871    elseif item.id == node.id'math' then
6872      inmath = (item.subtype == 0)
6873
6874    else
6875      d = nil
6876    end
6877
6878    -- AL <= EN/ET/ES    -- W2 + W3 + W6
```

```
6879    if last == 'al' and d == 'en' then
6880      d = 'an'              -- W3
6881    elseif last == 'al' and (d == 'et' or d == 'es') then
6882      d = 'on'              -- W6
6883    end
6884
6885    -- EN + CS/ES + EN      -- W4
6886    if d == 'en' and #nodes >= 2 then
6887      if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
6888          and nodes[#nodes-1][2] == 'en' then
6889        nodes[#nodes][2] = 'en'
6890      end
6891    end
6892
6893    -- AN + CS + AN          -- W4 too, because uax9 mixes both cases
6894    if d == 'an' and #nodes >= 2 then
6895      if (nodes[#nodes][2] == 'cs')
6896          and nodes[#nodes-1][2] == 'an' then
6897        nodes[#nodes][2] = 'an'
6898      end
6899    end
6900
6901    -- ET/EN                 -- W5 + W7->l / W6->on
6902    if d == 'et' then
6903      first_et = first_et or (#nodes + 1)
6904    elseif d == 'en' then
6905      has_en = true
6906      first_et = first_et or (#nodes + 1)
6907    elseif first_et then       -- d may be nil here !
6908      if has_en then
6909        if last == 'l' then
6910          temp = 'l'     -- W7
6911        else
6912          temp = 'en'    -- W5
6913        end
6914      else
6915        temp = 'on'      -- W6
6916      end
6917      for e = first_et, #nodes do
6918        if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
6919      end
6920      first_et = nil
6921      has_en = false
6922    end
6923
6924    -- Force mathdir in math if ON (currently works as expected only
6925    -- with 'l')
6926    if inmath and d == 'on' then
6927      d = ('TRT' == tex.mathdir) and 'r' or 'l'
6928    end
6929
6930    if d then
6931      if d == 'al' then
6932        d = 'r'
6933        last = 'al'
6934      elseif d == 'l' or d == 'r' then
6935        last = d
6936      end
6937      prev_d = d
```

```
6938        table.insert(nodes, {item, d, outer_first})
6939      end
6940
6941    outer_first = nil
6942
6943  end
6944
6945  -- TODO -- repeated here in case EN/ET is the last node. Find a
6946  -- better way of doing things:
6947  if first_et then        -- dir may be nil here !
6948    if has_en then
6949      if last == 'l' then
6950        temp = 'l'    -- W7
6951      else
6952        temp = 'en'   -- W5
6953      end
6954    else
6955      temp = 'on'      -- W6
6956    end
6957    for e = first_et, #nodes do
6958      if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
6959    end
6960  end
6961
6962  -- dummy node, to close things
6963  table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
6964
6965  --------------  NEUTRAL -----------------
6966
6967  outer = save_outer
6968  last = outer
6969
6970  local first_on = nil
6971
6972  for q = 1, #nodes do
6973    local item
6974
6975    local outer_first = nodes[q][3]
6976    outer = outer_first or outer
6977    last = outer_first or last
6978
6979    local d = nodes[q][2]
6980    if d == 'an' or d == 'en' then d = 'r' end
6981    if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
6982
6983    if d == 'on' then
6984      first_on = first_on or q
6985    elseif first_on then
6986      if last == d then
6987        temp = d
6988      else
6989        temp = outer
6990      end
6991      for r = first_on, q - 1 do
6992        nodes[r][2] = temp
6993        item = nodes[r][1]    -- MIRRORING
6994        if Babel.mirroring_enabled and item.id == GLYPH
6995            and temp == 'r' and characters[item.char] then
6996          local font_mode = font.fonts[item.font].properties.mode
```

```
6997          if font_mode ~= 'harf' and font_mode ~= 'plug' then
6998            item.char = characters[item.char].m or item.char
6999          end
7000        end
7001      end
7002      first_on = nil
7003    end

7005    if d == 'r' or d == 'l' then last = d end
7006  end

7008  -------------  IMPLICIT, REORDER ----------------

7010  outer = save_outer
7011  last = outer

7013  local state = {}
7014  state.has_r = false

7016  for q = 1, #nodes do

7018    local item = nodes[q][1]

7020    outer = nodes[q][3] or outer

7022    local d = nodes[q][2]

7024    if d == 'nsm' then d = last end               -- W1
7025    if d == 'en' then d = 'an' end
7026    local isdir = (d == 'r' or d == 'l')

7028    if outer == 'l' and d == 'an' then
7029      state.san = state.san or item
7030      state.ean = item
7031    elseif state.san then
7032      head, state = insert_numeric(head, state)
7033    end

7035    if outer == 'l' then
7036      if d == 'an' or d == 'r' then      -- im -> implicit
7037        if d == 'r' then state.has_r = true end
7038        state.sim = state.sim or item
7039        state.eim = item
7040      elseif d == 'l' and state.sim and state.has_r then
7041        head, state = insert_implicit(head, state, outer)
7042      elseif d == 'l' then
7043        state.sim, state.eim, state.has_r = nil, nil, false
7044      end
7045    else
7046      if d == 'an' or d == 'l' then
7047        if nodes[q][3] then -- nil except after an explicit dir
7048          state.sim = item  -- so we move sim 'inside' the group
7049        else
7050          state.sim = state.sim or item
7051        end
7052        state.eim = item
7053      elseif d == 'r' and state.sim then
7054        head, state = insert_implicit(head, state, outer)
7055      elseif d == 'r' then
```

211

```
7056        state.sim, state.eim = nil, nil
7057      end
7058    end
7059
7060    if isdir then
7061      last = d            -- Don't search back - best save now
7062    elseif d == 'on' and state.san  then
7063      state.san = state.san or item
7064      state.ean = item
7065    end
7066
7067  end
7068
7069  return node.prev(head) or head
7070 end
7071 ⟨/basic⟩
```

14   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.

15   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.

```
7072 ⟨*nil⟩
7073 \ProvidesLanguage{nil}[⟨⟨date⟩⟩ ⟨⟨version⟩⟩ Nil language]
7074 \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.

```
7075 \ifx\l@nil\@undefined
7076  \newlanguage\l@nil
7077  \@namedef{bbl@hyphendata@\the\l@nil}{{}{}}% Remove warning
7078  \let\bbl@elt\relax
7079  \edef\bbl@languages{%  Add it to the list of languages
7080    \bbl@languages\bbl@elt{nil}{\the\l@nil}{}{}}
7081 \fi
```

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

```
7082 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}
```

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

7083 \let\captionsnil\@empty
7084 \let\datenil\@empty

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.

7085 \ldf@finish{nil}
7086 ⟨/nil⟩

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

16.1   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.

7087 ⟨∗bplain | blplain⟩
7088 \catcode`\{=1 % left brace is begin-group character
7089 \catcode`\}=2 % right brace is end-group character
7090 \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).

7091 \openin 0 hyphen.cfg
7092 \ifeof0
7093 \else
7094   \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.

7095   \def\input #1 {%
7096   \let\input\a
7097   \a hyphen.cfg
7098   \let\a\undefined
7099   }
7100 \fi
7101 ⟨/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.

7102 ⟨bplain⟩\a plain.tex
7103 ⟨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.

```
7104 ⟨bplain⟩\def\fmtname{babel-plain}
7105 ⟨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.

16.2   Emulating some LaTeX features

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

```
7106 ⟨⟨*Emulate LaTeX⟩⟩ ≡
7107 \def\@empty{}
7108 \def\loadlocalcfg#1{%
7109   \openin0#1.cfg
7110   \ifeof0
7111     \closein0
7112   \else
7113     \closein0
7114     {\immediate\write16{************************************}%
7115     \immediate\write16{* Local config file #1.cfg used}%
7116     \immediate\write16{*}%
7117     }
7118     \input #1.cfg\relax
7119   \fi
7120   \@endofldf}
```

16.3   General tools

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

```
7121 \long\def\@firstofone#1{#1}
7122 \long\def\@firstoftwo#1#2{#1}
7123 \long\def\@secondoftwo#1#2{#2}
7124 \def\@nnil{\@nil}
7125 \def\@gobbletwo#1#2{}
7126 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
7127 \def\@star@or@long#1{%
7128   \@ifstar
7129   {\let\l@ngrel@x\relax#1}%
7130   {\let\l@ngrel@x\long#1}}
7131 \let\l@ngrel@x\relax
7132 \def\@car#1#2\@nil{#1}
7133 \def\@cdr#1#2\@nil{#2}
7134 \let\@typeset@protect\relax
7135 \let\protected@edef\edef
7136 \long\def\@gobble#1{}
7137 \edef\@backslashchar{\expandafter\@gobble\string\\}
7138 \def\strip@prefix#1>{}
7139 \def\g@addto@macro#1#2{{%
7140     \toks@\expandafter{#1#2}%
7141     \xdef#1{\the\toks@}}}
7142 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
7143 \def\@nameuse#1{\csname #1\endcsname}
7144 \def\@ifundefined#1{%
```

```
7145    \expandafter\ifx\csname#1\endcsname\relax
7146      \expandafter\@firstoftwo
7147    \else
7148      \expandafter\@secondoftwo
7149    \fi}
7150 \def\@expandtwoargs#1#2#3{%
7151    \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
7152 \def\zap@space#1 #2{%
7153    #1%
7154    \ifx#2\@empty\else\expandafter\zap@space\fi
7155    #2}
7156 \let\bbl@trace\@gobble
7157 \def\bbl@error#1#2{%
7158    \begingroup
7159      \newlinechar=`\^^J
7160      \def\\{^^J(babel) }%
7161      \errhelp{#2}\errmessage{\\#1}%
7162    \endgroup}
7163 \def\bbl@warning#1{%
7164    \begingroup
7165      \newlinechar=`\^^J
7166      \def\\{^^J(babel) }%
7167      \message{\\#1}%
7168    \endgroup}
7169 \let\bbl@infowarn\bbl@warning
7170 \def\bbl@info#1{%
7171    \begingroup
7172      \newlinechar=`\^^J
7173      \def\\{^^J}%
7174      \wlog{#1}%
7175    \endgroup}
```

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

```
7176 \ifx\@preamblecmds\@undefined
7177    \def\@preamblecmds{}
7178 \fi
7179 \def\@onlypreamble#1{%
7180    \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
7181      \@preamblecmds\do#1}}
7182 \@onlypreamble\@onlypreamble
```

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

```
7183 \def\begindocument{%
7184    \@begindocumenthook
7185    \global\let\@begindocumenthook\@undefined
7186    \def\do##1{\global\let##1\@undefined}%
7187    \@preamblecmds
7188    \global\let\do\noexpand}

7189 \ifx\@begindocumenthook\@undefined
7190    \def\@begindocumenthook{}
7191 \fi
7192 \@onlypreamble\@begindocumenthook
7193 \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.

```
7194 \def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
7195 \@onlypreamble\AtEndOfPackage
```

```
7196 \def\@endofldf{}
7197 \@onlypreamble\@endofldf
7198 \let\bbl@afterlang\@empty
7199 \chardef\bbl@opt@hyphenmap\z@
```

LᴬTᴇX 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.

```
7200 \catcode`\&=\z@
7201 \ifx&if@filesw\@undefined
7202   \expandafter\let\csname if@filesw\expandafter\endcsname
7203     \csname iffalse\endcsname
7204 \fi
7205 \catcode`\&=4
```

Mimick LᴬTᴇX's commands to define control sequences.

```
7206 \def\newcommand{\@star@or@long\new@command}
7207 \def\new@command#1{%
7208   \@testopt{\@newcommand#1}0}
7209 \def\@newcommand#1[#2]{%
7210   \@ifnextchar [{\@xargdef#1[#2]}%
7211                 {\@argdef#1[#2]}}
7212 \long\def\@argdef#1[#2]#3{%
7213   \@yargdef#1\@ne{#2}{#3}}
7214 \long\def\@xargdef#1[#2][#3]#4{%
7215   \expandafter\def\expandafter#1\expandafter{%
7216     \expandafter\@protected@testopt\expandafter #1%
7217     \csname\string#1\expandafter\endcsname{#3}}%
7218   \expandafter\@yargdef \csname\string#1\endcsname
7219   \tw@{#2}{#4}}
7220 \long\def\@yargdef#1#2#3{%
7221   \@tempcnta#3\relax
7222   \advance \@tempcnta \@ne
7223   \let\@hash@\relax
7224   \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
7225   \@tempcntb #2%
7226   \@whilenum\@tempcntb <\@tempcnta
7227   \do{%
7228     \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
7229     \advance\@tempcntb \@ne}%
7230   \let\@hash@##%
7231   \l@ngrel@x\expandafter\def\expandafter#1\reserved@a}
7232 \def\providecommand{\@star@or@long\provide@command}
7233 \def\provide@command#1{%
7234   \begingroup
7235     \escapechar\m@ne\xdef\@gtempa{{\string#1}}%
7236   \endgroup
7237   \expandafter\@ifundefined\@gtempa
7238     {\def\reserved@a{\new@command#1}}%
7239     {\let\reserved@a\relax
7240      \def\reserved@a{\new@command\reserved@a}}%
7241   \reserved@a}%
7242 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
7243 \def\declare@robustcommand#1{%
7244   \edef\reserved@a{\string#1}%
7245   \def\reserved@b{#1}%
7246   \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
7247   \edef#1{%
7248     \ifx\reserved@a\reserved@b
```

```
7249        \noexpand\x@protect
7250        \noexpand#1%
7251      \fi
7252      \noexpand\protect
7253      \expandafter\noexpand\csname
7254        \expandafter\@gobble\string#1 \endcsname
7255    }%
7256    \expandafter\new@command\csname
7257      \expandafter\@gobble\string#1 \endcsname
7258 }
7259 \def\x@protect#1{%
7260    \ifx\protect\@typeset@protect\else
7261      \@x@protect#1%
7262    \fi
7263 }
7264 \catcode`\&=\z@  % Trick to hide conditionals
7265    \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.

```
7266    \def\bbl@tempa{\csname newif\endcsname&ifin@}
7267 \catcode`\&=4
7268 \ifx\in@\@undefined
7269   \def\in@#1#2{%
7270     \def\in@@##1#1##2##3\in@@{%
7271       \ifx\in@##2\in@false\else\in@true\fi}%
7272     \in@@#2#1\in@\in@@}
7273 \else
7274   \let\bbl@tempa\@empty
7275 \fi
7276 \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).

```
7277 \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.

```
7278 \def\@ifl@aded#1#2#3#4{}
```

For the following code we need to make sure that the commands \newcommand and \providecommand exist with some sensible definition. They are not fully equivalent to their LaTeX 2ε versions; just enough to make things work in plain TeXenvironments.

```
7279 \ifx\@tempcnta\@undefined
7280   \csname newcount\endcsname\@tempcnta\relax
7281 \fi
7282 \ifx\@tempcntb\@undefined
7283   \csname newcount\endcsname\@tempcntb\relax
7284 \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).

```
7285 \ifx\bye\@undefined
7286   \advance\count10 by -2\relax
7287 \fi
7288 \ifx\@ifnextchar\@undefined
```

```
7289   \def\@ifnextchar#1#2#3{%
7290     \let\reserved@d=#1%
7291     \def\reserved@a{#2}\def\reserved@b{#3}%
7292     \futurelet\@let@token\@ifnch}
7293   \def\@ifnch{%
7294     \ifx\@let@token\@sptoken
7295       \let\reserved@c\@xifnch
7296     \else
7297       \ifx\@let@token\reserved@d
7298         \let\reserved@c\reserved@a
7299       \else
7300         \let\reserved@c\reserved@b
7301       \fi
7302     \fi
7303     \reserved@c}
7304   \def\:{\let\@sptoken= } \:  % this makes \@sptoken a space token
7305   \def\:{\@xifnch} \expandafter\def\: {\futurelet\@let@token\@ifnch}
7306 \fi
7307 \def\@testopt#1#2{%
7308   \@ifnextchar[{#1}{#1[{#2}]}}
7309 \def\@protected@testopt#1{%
7310   \ifx\protect\@typeset@protect
7311     \expandafter\@testopt
7312   \else
7313     \@x@protect#1%
7314   \fi}
7315 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
7316       #2\relax}\fi}
7317 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
7318         \else\expandafter\@gobble\fi{#1}}
```

16.4 Encoding related macros

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

```
7319 \def\DeclareTextCommand{%
7320   \@dec@text@cmd\providecommand
7321 }
7322 \def\ProvideTextCommand{%
7323   \@dec@text@cmd\providecommand
7324 }
7325 \def\DeclareTextSymbol#1#2#3{%
7326   \@dec@text@cmd\chardef#1{#2}#3\relax
7327 }
7328 \def\@dec@text@cmd#1#2#3{%
7329   \expandafter\def\expandafter#2%
7330     \expandafter{%
7331       \csname#3-cmd\expandafter\endcsname
7332       \expandafter#2%
7333       \csname#3\string#2\endcsname
7334     }%
7335 %   \let\@ifdefinable\@rc@ifdefinable
7336   \expandafter#1\csname#3\string#2\endcsname
7337 }
7338 \def\@current@cmd#1{%
7339   \ifx\protect\@typeset@protect\else
7340     \noexpand#1\expandafter\@gobble
7341   \fi
7342 }
```

```
7343 \def\@changed@cmd#1#2{%
7344    \ifx\protect\@typeset@protect
7345       \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
7346          \expandafter\ifx\csname ?\string#1\endcsname\relax
7347             \expandafter\def\csname ?\string#1\endcsname{%
7348                \@changed@x@err{#1}%
7349             }%
7350          \fi
7351          \global\expandafter\let
7352             \csname\cf@encoding \string#1\expandafter\endcsname
7353             \csname ?\string#1\endcsname
7354       \fi
7355       \csname\cf@encoding\string#1%
7356          \expandafter\endcsname
7357    \else
7358       \noexpand#1%
7359    \fi
7360 }
7361 \def\@changed@x@err#1{%
7362    \errhelp{Your command will be ignored, type <return> to proceed}%
7363    \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
7364 \def\DeclareTextCommandDefault#1{%
7365    \DeclareTextCommand#1?%
7366 }
7367 \def\ProvideTextCommandDefault#1{%
7368    \ProvideTextCommand#1?%
7369 }
7370 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
7371 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
7372 \def\DeclareTextAccent#1#2#3{%
7373   \DeclareTextCommand#1{#2}[1]{\accent#3 ##1}
7374 }
7375 \def\DeclareTextCompositeCommand#1#2#3#4{%
7376    \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
7377    \edef\reserved@b{\string##1}%
7378    \edef\reserved@c{%
7379      \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
7380    \ifx\reserved@b\reserved@c
7381       \expandafter\expandafter\expandafter\ifx
7382          \expandafter\@car\reserved@a\relax\relax\@nil
7383          \@text@composite
7384       \else
7385          \edef\reserved@b##1{%
7386             \def\expandafter\noexpand
7387                \csname#2\string#1\endcsname####1{%
7388                \noexpand\@text@composite
7389                   \expandafter\noexpand\csname#2\string#1\endcsname
7390                   ####1\noexpand\@empty\noexpand\@text@composite
7391                   {##1}%
7392             }%
7393          }%
7394          \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
7395       \fi
7396       \expandafter\def\csname\expandafter\string\csname
7397          #2\endcsname\string#1-\string#3\endcsname{#4}
7398    \else
7399       \errhelp{Your command will be ignored, type <return> to proceed}%
7400       \errmessage{\string\DeclareTextCompositeCommand\space used on
7401          inappropriate command \protect#1}
```

219

```
7402    \fi
7403 }
7404 \def\@text@composite#1#2#3\@text@composite{%
7405    \expandafter\@text@composite@x
7406       \csname\string#1-\string#2\endcsname
7407 }
7408 \def\@text@composite@x#1#2{%
7409    \ifx#1\relax
7410       #2%
7411    \else
7412       #1%
7413    \fi
7414 }
7415 %
7416 \def\@strip@args#1:#2-#3\@strip@args{#2}
7417 \def\DeclareTextComposite#1#2#3#4{%
7418    \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
7419    \bgroup
7420       \lccode`\@=#4%
7421       \lowercase{%
7422    \egroup
7423       \reserved@a @%
7424    }%
7425 }
7426 %
7427 \def\UseTextSymbol#1#2{#2}
7428 \def\UseTextAccent#1#2#3{}
7429 \def\@use@text@encoding#1{}
7430 \def\DeclareTextSymbolDefault#1#2{%
7431    \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
7432 }
7433 \def\DeclareTextAccentDefault#1#2{%
7434    \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
7435 }
7436 \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.

```
7437 \DeclareTextAccent{\"}{OT1}{127}
7438 \DeclareTextAccent{\'}{OT1}{19}
7439 \DeclareTextAccent{\^}{OT1}{94}
7440 \DeclareTextAccent{\`}{OT1}{18}
7441 \DeclareTextAccent{\~}{OT1}{126}
```

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

```
7442 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
7443 \DeclareTextSymbol{\textquotedblright}{OT1}{`\"}
7444 \DeclareTextSymbol{\textquoteleft}{OT1}{`\`}
7445 \DeclareTextSymbol{\textquoteright}{OT1}{`\'}
7446 \DeclareTextSymbol{\i}{OT1}{16}
7447 \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.

```
7448 \ifx\scriptsize\@undefined
7449   \let\scriptsize\sevenrm
7450 \fi
```

And a few more "dummy" definitions.

```
7451 \def\languagename{english}%
7452 \let\bbl@opt@shorthands\@nnil
7453 \def\bbl@ifshorthand#1#2#3{#2}%
7454 \let\bbl@language@opts\@empty
7455 \ifx\babeloptionstrings\@undefined
7456   \let\bbl@opt@strings\@nnil
7457 \else
7458   \let\bbl@opt@strings\babeloptionstrings
7459 \fi
7460 \def\BabelStringsDefault{generic}
7461 \def\bbl@tempa{normal}
7462 \ifx\babeloptionmath\bbl@tempa
7463   \def\bbl@mathnormal{\noexpand\textormath}
7464 \fi
7465 \def\AfterBabelLanguage#1#2{}
7466 \ifx\BabelModifiers\@undefined\let\BabelModifiers\relax\fi
7467 \let\bbl@afterlang\relax
7468 \def\bbl@opt@safe{BR}
7469 \ifx\@uclclist\@undefined\let\@uclclist\@empty\fi
7470 \ifx\bbl@trace\@undefined\def\bbl@trace#1{}\fi
7471 \expandafter\newif\csname ifbbl@single\endcsname
7472 \chardef\bbl@bidimode\z@
7473 ⟨⟨/Emulate LaTeX⟩⟩
```

A proxy file:

```
7474 ⟨*plain⟩
7475 \input babel.def
7476 ⟨/plain⟩
```

17 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).