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.

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

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

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

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

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

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

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

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

1.2   Multilingual documents

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

EXAMPLE In LaTeX, the preamble of the document:

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

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

You can also set the main language explicitly, but it is discouraged except if there 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).

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

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

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

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

```
\babelensure{polish}
```

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

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

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

1.10 Shorthands

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

NOTE Keep in mind the following:

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

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

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

TROUBLESHOOTING A typical error when using shorthands is the following:

---

[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.

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]{1ດ 1ຍ 1ຣ 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

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]		
seh	Sena	zh-Hans-HK	Chinese
ses	Koyraboro Senni	zh-Hans-MO	Chinese
sg	Sango	zh-Hans-SG	Chinese
shi-Latn	Tachelhit	zh-Hans	Chinese
shi-Tfng	Tachelhit	zh-Hant-HK	Chinese

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	burmese
akan	canadian
albanian	cantonese
american	catalan
amharic	centralatlastamazight
ancientgreek	centralkurdish
arabic	chechen
arabic-algeria	cherokee
arabic-DZ	chiga
arabic-morocco	chinese-hans-hk
arabic-MA	chinese-hans-mo
arabic-syria	chinese-hans-sg
arabic-SY	chinese-hans
armenian	chinese-hant-hk
assamese	chinese-hant-mo
asturian	chinese-hant
asu	chinese-simplified-hongkongsarchina
australian	chinese-simplified-macausarchina
austrian	chinese-simplified-singapore
azerbaijani-cyrillic	chinese-simplified
azerbaijani-cyrl	chinese-traditional-hongkongsarchina
azerbaijani-latin	chinese-traditional-macausarchina
azerbaijani-latn	chinese-traditional
azerbaijani	chinese
bafia	churchslavic
bambara	churchslavic-cyrs
basaa	churchslavic-oldcyrillic[12]
basque	churchsslavic-glag
belarusian	churchsslavic-glagolitic
bemba	colognian
bena	cornish
bengali	croatian
bodo	czech
bosnian-cyrillic	danish
bosnian-cyrl	duala
bosnian-latin	dutch
bosnian-latn	dzongkha
bosnian	embu
brazilian	english-au
breton	english-australia
british	english-ca
bulgarian	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.

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}%
}
```

28

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

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

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

This redefinition is immediate.

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

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

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

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

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

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

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

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

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

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

1.16   Creating a language

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

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

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

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

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

EXAMPLE If you need a language named `arhinish`:

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

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

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

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

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

import=   ⟨*language-tag*⟩

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

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

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

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

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

30

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

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

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

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 ìni 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.

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

---

[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.

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}`.
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.
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.

---

[15]They are similar in concept, but not the same, as those in Unicode.

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 {⟨*hyphenrules-name*⟩}{⟨*lua-pattern*⟩}{⟨*replacement*⟩}

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

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

In the replacements, a captured char may be mapped to another, too. For example, if the first capture reads ([ǐǔ]), the replacement could be {1|ǐǔ|íú}, which maps ǐ to í, and ǔ to ú, so that the diaeresis is removed.

This feature is activated with the first \babelposthyphenation or \babelprehyphenation. 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` $\rightarrow$ `fr-Latn` $\rightarrow$ `fr-FR` $\rightarrow$ `fr`.
Languages with the same resolved name are considered the same. Case is normalized
before, so that `fr-latn-fr` $\rightarrow$ `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

---

[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.

41

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

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 العصر فصحى \textit{fuṣḥā l-‘aṣr} (MSA) and
التراث فصحى \textit{fuṣḥā t-turāth} (CA).

\end{document}
```

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

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).

`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:

---

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

```
    \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}{}{}{}
```

45

(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.
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.

---

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

1.28 Unicode character properties in luatex

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

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

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

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

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

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

1.29 Tweaking some features

\babeladjust {⟨*key-value-list*⟩}

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

1.30 Tips, workarounds, known issues and notes

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

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

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

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

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

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

- For the hyphenation to work correctly, lccodes cannot change, because TeX only takes into account the values when the paragraph is hyphenated, i.e., when it has been finished.[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.
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.

---

[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.

[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.

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).

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:

---

[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.

```
% 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.

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

---

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

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 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.

---

[26]But not removed, for backward compatibility.

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

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

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

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

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

3.2 Basic macros

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

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

`\adddialect` The macro `\adddialect` can be used when two languages can (or must) use the same hyphenation patterns. This can also be useful for languages for which no patterns are preloaded in the format. In such cases the default behavior of the babel system is to define this language as a 'dialect' of the language for which the patterns were loaded as `\language0`. Here "language" is used in the TeX sense of set of hyphenation patterns.

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

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

(Assigning `\lefthyphenmin` and `\righthyphenmin` directly in `\extras<lang>` has no effect.)

`\providehyphenmins` The macro `\providehyphenmins` should be used in the language definition files to set `\lefthyphenmin` and `\righthyphenmin`. This macro will check whether these parameters were provided by the hyphenation file before it takes any action. If these values have been already set, this command is ignored (currently, default pattern files do *not* set them).

`\captions⟨lang⟩` The macro `\captions⟨lang⟩` defines the macros that hold the texts to replace the original hard-wired texts.

`\date⟨lang⟩` The macro `\date⟨lang⟩` defines `\today`.

`\extras⟨lang⟩` The macro `\extras⟨lang⟩` contains all the extra definitions needed for a specific language. This macro, like the following, is a hook – you can add things to it, but it must not be used directly.

`\noextras⟨lang⟩` Because we want to let the user switch between languages, but we do not know what state TeX might be in after the execution of `\extras⟨lang⟩`, a macro that brings TeX into a predefined state is needed. It will be no surprise that the name of this macro is `\noextras⟨lang⟩`.

\bbl@declare@ttribute	This is a command to be used in the language definition files for declaring a language attribute. It takes three arguments: the name of the language, the attribute to be defined, and the code to be executed when the attribute is to be used.
\main@language	To postpone the activation of the definitions needed for a language until the beginning of a document, all language definition files should use \main@language instead of \selectlanguage. This will just store the name of the language, and the proper language will be activated at the start of the document.
\ProvidesLanguage	The macro \ProvidesLanguage should be used to identify the language definition files. Its syntax is similar to the syntax of the LaTeX command \ProvidesPackage.
\LdfInit	The macro \LdfInit performs a couple of standard checks that must be made at the beginning of a language definition file, such as checking the category code of the @-sign, preventing the .ldf file from being processed twice, etc.
\ldf@quit	The macro \ldf@quit does work needed if a .ldf file was processed earlier. This includes resetting the category code of the @-sign, preparing the language to be activated at \begin{document} time, and ending the input stream.
\ldf@finish	The macro \ldf@finish does work needed at the end of each .ldf file. This includes resetting the category code of the @-sign, loading a local configuration file, and preparing the language to be activated at \begin{document} time.
\loadlocalcfg	After processing a language definition file, LaTeX can be instructed to load a local configuration file. This file can, for instance, be used to add strings to \captions⟨*lang*⟩ to support local document classes. The user will be informed that this configuration file has been loaded. This macro is called by \ldf@finish.
\substitutefontfamily	(Deprecated.) This command takes three arguments, a font encoding and two font family names. It creates a font description file for the first font in the given encoding. This .fd file will instruct LaTeX to use a font from the second family when a font from the first family in the given encoding seems to be needed.

3.3 Skeleton

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

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

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

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

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

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

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

\StartBabelCommands*{<language>}{date}
```

```
\SetString\monthiname{<name of first month>}
% More strings

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

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

\EndBabelCommands

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

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

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

```
\AtEndOfPackage{%
  \RequirePackage{dingbat}%        Delay package
  \savebox{\myeye}{\eye}}%         And direct usage
\newsavebox{\myeye}
\newcommand\myanchor{\anchor}%     But OK inside command
```

3.4   Support for active characters

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

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

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

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

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

3.5 Support for saving macro definitions

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

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

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

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

3.6 Support for extending macros

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

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

3.7 Macros common to a number of languages

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

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

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

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

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

\bbl@frenchspacing   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

---

[27]This mechanism was introduced by Bernd Raichle.

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:

---

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

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

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

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

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

\StartBabelCommands{german,austrian}{date}
  \SetString\monthiiname{Februar}
  \SetString\monthiiiname{M\"{a}rz}
  \SetString\monthivname{April}
  \SetString\monthvname{Mai}
  \SetString\monthviname{Juni}
  \SetString\monthviiname{Juli}
  \SetString\monthviiiname{August}
  \SetString\monthixname{September}
  \SetString\monthxname{Oktober}
  \SetString\monthxiname{November}
  \SetString\monthxiiname{Dezenber}
  \SetString\today{\number\day.~%
    \csname month\romannumeral\month name\endcsname\space
    \number\year}

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

\EndBabelCommands
```

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

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

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

\EndBabelCommands   Marks the end of the series of blocks.

\AfterBabelCommands   {⟨*code*⟩}

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

---

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

59

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

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

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

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

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

#1 is replaced by the roman numeral.

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

Sets globally code to be executed at \MakeUppercase and \MakeLowercase. The code would typically be things like \let\BB\bb and \uccode or \lccode (although for the reasons explained above, changes in lc/uc codes may not work). A ⟨*map-list*⟩ is a series of macros using the internal format of \@uclclist (eg, \bb\BB\cc\CC). The mandatory arguments take precedence over the optional one. This command, unlike \SetString, is executed always (even without `strings`), and it is intended for minor readjustments only. For example, as T1 is the default case mapping in LaTeX, we can set for Turkish:

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

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

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

\EndBabelCommands
```

(Note the mapping for OT1 is not complete.)

\SetHyphenMap   {⟨*to-lower-macros*⟩}

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

There are three helper macros to be used inside \SetHyphenMap:

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

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

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

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

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

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

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

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}%
```

---

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

```
35    \bbl@exp@aux}
36 \def\bbl@exp@en#1>{\expandafter\noexpand\csname#1\endcsname}%
37 \def\bbl@exp@ue#1]{%
38    \unexpanded\expandafter\expandafter\expandafter{\csname#1\endcsname}}%
```

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

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

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

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

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

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

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*{%
359   \bbl@xin@{\string=}{\CurrentOption}%
360   \ifin@
361     \expandafter\bbl@tempa\CurrentOption\bbl@tempa
362   \else
363     \bbl@xin@{,\CurrentOption,}{,\bbl@language@opts,}%
364     \ifin@
365       \bbl@exp{\\\bbl@replace\\\bbl@language@opts{,\CurrentOption,}{}}%
366     \fi
367     \edef\bbl@language@opts{\bbl@language@opts,\CurrentOption,}
368   \fi}
```

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

```
369 \ProcessOptions*
```

```
370 \ifx\bbl@opt@provide\@nnil
371   \let\bbl@opt@provide\@empty  % %%% MOVE above
372 \else
373   \chardef\bbl@iniflag\@ne
374   \bbl@exp{\\\bbl@forkv{\@nameuse{@raw@opt@babel.sty}}}{%
375     \in@{,provide,}{,#1,}%
376     \ifin@
377       \def\bbl@opt@provide{#2}%
378       \bbl@replace\bbl@opt@provide{;}{,}%
379     \fi}
380 \fi
381 %
```

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=....

```
382 \bbl@trace{Conditional loading of shorthands}
383 \def\bbl@sh@string#1{%
384   \ifx#1\@empty\else
385     \ifx#1t\string~%
```

71

```
386    \else\ifx#1c\string,%
387    \else\string#1%
388    \fi\fi
389    \expandafter\bbl@sh@string
390    \fi}
391 \ifx\bbl@opt@shorthands\@nnil
392   \def\bbl@ifshorthand#1#2#3{#2}%
393 \else\ifx\bbl@opt@shorthands\@empty
394   \def\bbl@ifshorthand#1#2#3{#3}%
395 \else
```

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

```
396   \def\bbl@ifshorthand#1{%
397     \bbl@xin@{\string#1}{\bbl@opt@shorthands}%
398     \ifin@
399       \expandafter\@firstoftwo
400     \else
401       \expandafter\@secondoftwo
402     \fi}
```

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

```
403   \edef\bbl@opt@shorthands{%
404     \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.

```
405   \bbl@ifshorthand{'}%
406     {\PassOptionsToPackage{activeacute}{babel}}{}
407   \bbl@ifshorthand{`}%
408     {\PassOptionsToPackage{activegrave}{babel}}{}
409 \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.

```
410 \ifx\bbl@opt@headfoot\@nnil\else
411   \g@addto@macro\@resetactivechars{%
412     \set@typeset@protect
413     \expandafter\select@language@x\expandafter{\bbl@opt@headfoot}%
414     \let\protect\noexpand}
415 \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.

```
416 \ifx\bbl@opt@safe\@undefined
417   \def\bbl@opt@safe{BR}
418 \fi
```

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

```
419 \ifx\bbl@opt@main\@nnil\else
420   \edef\bbl@language@opts{\bbl@language@opts,\bbl@opt@main,}
421 \fi
```

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

```
422 \bbl@trace{Defining IfBabelLayout}
423 \ifx\bbl@opt@layout\@nnil
424   \newcommand\IfBabelLayout[3]{#3}%
425 \else
426   \newcommand\IfBabelLayout[1]{%
```

```
427    \@expandtwoargs\in@{.#1.}{.\bbl@opt@layout.}%
428    \ifin@
429      \expandafter\@firstoftwo
430    \else
431      \expandafter\@secondoftwo
432    \fi}
433 \fi
434 ⟨/package⟩
435 ⟨*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*.

```
436 \ifx\ldf@quit\@undefined\else
437 \endinput\fi % Same line!
438 ⟨⟨Make sure ProvidesFile is defined⟩⟩
439 \ProvidesFile{babel.def}[⟨⟨date⟩⟩ ⟨⟨version⟩⟩ Babel common definitions]
440 \ifx\AtBeginDocument\@undefined  % TODO. change test.
441    ⟨⟨Emulate LaTeX⟩⟩
442 \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.

```
443 ⟨/core⟩
444 ⟨*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.

```
445 \def\bbl@version{⟨⟨version⟩⟩}
446 \def\bbl@date{⟨⟨date⟩⟩}
447 ⟨⟨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.

```
448 \def\adddialect#1#2{%
449   \global\chardef#1#2\relax
450   \bbl@usehooks{adddialect}{{#1}{#2}}%
451   \begingroup
452     \count@#1\relax
453     \def\bbl@elt##1##2##3##4{%
454       \ifnum\count@=##2\relax
455         \edef\bbl@tempa{\expandafter\@gobbletwo\string#1}%
456         \bbl@info{Hyphen rules for '\expandafter\@gobble\bbl@tempa'
457                 set to \expandafter\string\csname l@##1\endcsname\\%
458                 (\string\language\the\count@). Reported}%
459         \def\bbl@elt####1####2####3####4{}%
460       \fi}%
461     \bbl@cs{languages}%
462   \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.

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

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

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

```
511      \fi
512      \ifx\bbl@bcp\relax
513        \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
514          {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
515          {}%
516      \fi
517      \ifx\bbl@bcp\relax
518        \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
519      \fi
520    \fi\fi}
521 \let\bbl@initoload\relax
522 \def\bbl@provide@locale{%
523    \ifx\babelprovide\@undefined
524      \bbl@error{For a language to be defined on the fly 'base'\\%
525                  is not enough, and the whole package must be\\%
526                  loaded. Either delete the 'base' option or\\%
527                  request the languages explicitly}%
528                 {See the manual for further details.}%
529    \fi
530 % TODO. Option to search if loaded, with \LocaleForEach
531    \let\bbl@auxname\languagename % Still necessary. TODO
532    \bbl@ifunset{bbl@bcp@map@\languagename}{}% Move uplevel??
533      {\edef\languagename{\@nameuse{bbl@bcp@map@\languagename}}}%
534    \ifbbl@bcpallowed
535      \expandafter\ifx\csname date\languagename\endcsname\relax
536        \expandafter
537        \bbl@bcplookup\languagename-\@empty-\@empty-\@empty\@@
538        \ifx\bbl@bcp\relax\else  % Returned by \bbl@bcplookup
539          \edef\languagename{\bbl@bcp@prefix\bbl@bcp}%
540          \edef\localename{\bbl@bcp@prefix\bbl@bcp}%
541          \expandafter\ifx\csname date\languagename\endcsname\relax
542            \let\bbl@initoload\bbl@bcp
543            \bbl@exp{\\\babelprovide[\bbl@autoload@bcpoptions]{\languagename}}%
544            \let\bbl@initoload\relax
545          \fi
546          \bbl@csarg\xdef{bcp@map@\bbl@bcp}{\localename}%
547        \fi
548      \fi
549    \fi
550    \expandafter\ifx\csname date\languagename\endcsname\relax
551      \IfFileExists{babel-\languagename.tex}%
552        {\bbl@exp{\\\babelprovide[\bbl@autoload@options]{\languagename}}}%
553        {}%
554    \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.

```
555 \def\iflanguage#1{%
556    \bbl@iflanguage{#1}{%
557      \ifnum\csname l@#1\endcsname=\language
558        \expandafter\@firstoftwo
559      \else
560        \expandafter\@secondoftwo
561      \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.

```
562 \let\bbl@select@type\z@
563 \edef\selectlanguage{%
564   \noexpand\protect
565   \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.

```
566 \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.

```
567 \let\xstring\string
```

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

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

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

```
568 \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

```
569 \def\bbl@push@language{%
570   \ifx\languagename\@undefined\else
571     \ifx\currentgrouplevel\@undefined
572       \xdef\bbl@language@stack{\languagename+\bbl@language@stack}%
573     \else
574       \ifnum\currentgrouplevel=\z@
575         \xdef\bbl@language@stack{\languagename+}%
576       \else
577         \xdef\bbl@language@stack{\languagename+\bbl@language@stack}%
578       \fi
579     \fi
580   \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.

```
581 \def\bbl@pop@lang#1+#2\@@{%
582   \edef\languagename{#1}%
583   \xdef\bbl@language@stack{#2}}
```

76

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).

```
584 \let\bbl@ifrestoring\@secondoftwo
585 \def\bbl@pop@language{%
586   \expandafter\bbl@pop@lang\bbl@language@stack\@@
587   \let\bbl@ifrestoring\@firstoftwo
588   \expandafter\bbl@set@language\expandafter{\languagename}%
589   \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).

```
590 \chardef\localeid\z@
591 \def\bbl@id@last{0}    % No real need for a new counter
592 \def\bbl@id@assign{%
593   \bbl@ifunset{bbl@id@@\languagename}%
594     {\count@\bbl@id@last\relax
595      \advance\count@\@ne
596      \bbl@csarg\chardef{id@@\languagename}\count@
597      \edef\bbl@id@last{\the\count@}%
598      \ifcase\bbl@engine\or
599        \directlua{
600          Babel = Babel or {}
601          Babel.locale_props = Babel.locale_props or {}
602          Babel.locale_props[\bbl@id@last] = {}
603          Babel.locale_props[\bbl@id@last].name = '\languagename'
604        }%
605      \fi}%
606     {}%
607   \chardef\localeid\bbl@cl{id@}}
```

The unprotected part of \selectlanguage.

```
608 \expandafter\def\csname selectlanguage \endcsname#1{%
609   \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\tw@\fi
610   \bbl@push@language
611   \aftergroup\bbl@pop@language
612   \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).

```
613 \def\BabelContentsFiles{toc,lof,lot}
614 \def\bbl@set@language#1{% from selectlanguage, pop@
```

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

```
674      \else
675        % set type
676        \let\bbl@select@type\z@
677        \expandafter\bbl@switch\expandafter{\languagename}%
678      \fi}}
679 \def\babel@aux#1#2{%
680    \select@language{#1}%
681    \bbl@foreach\BabelContentsFiles{% \relax -> don't assume vertical mode
682      \@writefile{##1}{\babel@toc{#1}{#2}\relax}}}% TODO - plain?
683 \def\babel@toc#1#2{%
684    \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.

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

```
720    \bbl@usehooks{afterextras}{}%
721    %  > babel-ensure
722    %  > babel-sh-<short>
723    %  > babel-bidi
724    %  > babel-fontspec
725    % hyphenation - case mapping
726    \ifcase\bbl@opt@hyphenmap\or
727      \def\BabelLower##1##2{\lccode##1=##2\relax}%
728      \ifnum\bbl@hymapsel>4\else
729        \csname\languagename @bbl@hyphenmap\endcsname
730      \fi
731      \chardef\bbl@opt@hyphenmap\z@
732    \else
733      \ifnum\bbl@hymapsel>\bbl@opt@hyphenmap\else
734        \csname\languagename @bbl@hyphenmap\endcsname
735      \fi
736    \fi
737    \let\bbl@hymapsel\@cclv
738    % hyphenation - select rules
739    \ifnum\csname l@\languagename\endcsname=\l@unhyphenated
740      \edef\bbl@tempa{u}%
741    \else
742      \edef\bbl@tempa{\bbl@cl{lnbrk}}%
743    \fi
744    % linebreaking - handle u, e, k (v in the future)
745    \bbl@xin@{/u}{/\bbl@tempa}%
746    \ifin@\else\bbl@xin@{/e}{/\bbl@tempa}\fi % elongated forms
747    \ifin@\else\bbl@xin@{/k}{/\bbl@tempa}\fi % only kashida
748    \ifin@\else\bbl@xin@{/v}{/\bbl@tempa}\fi % variable font
749    \ifin@
750      % unhyphenated/kashida/elongated = allow stretching
751      \language\l@unhyphenated
752      \babel@savevariable\emergencystretch
753      \emergencystretch\maxdimen
754      \babel@savevariable\hbadness
755      \hbadness\@M
756    \else
757      % other = select patterns
758      \bbl@patterns{#1}%
759    \fi
760    % hyphenation - mins
761    \babel@savevariable\lefthyphenmin
762    \babel@savevariable\righthyphenmin
763    \expandafter\ifx\csname #1hyphenmins\endcsname\relax
764      \set@hyphenmins\tw@\thr@@\relax
765    \else
766      \expandafter\expandafter\expandafter\set@hyphenmins
767        \csname #1hyphenmins\endcsname\relax
768    \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.

```
769 \long\def\otherlanguage#1{%
770   \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\thr@@\fi
771   \csname selectlanguage \endcsname{#1}%
```

```
772    \ignorespaces}
```

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

```
773 \long\def\endotherlanguage{%
774    \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.

```
775 \expandafter\def\csname otherlanguage*\endcsname{%
776    \@ifnextchar[\bbl@otherlanguage@s{\bbl@otherlanguage@s[]}}
777 \def\bbl@otherlanguage@s[#1]#2{%
778    \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
779    \def\bbl@select@opts{#1}%
780    \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".

```
781 \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.

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

```
799      {\par}%
800      \let\bbl@select@opts\@empty
801      \let\BabelText\@firstofone
802      \foreign@language{#1}%
803      \bbl@usehooks{foreign*}{}%
804      \bbl@dirparastext
805      \BabelText{#2}% Still in vertical mode!
806      {\par}%
807    \endgroup}
```

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

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

\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.

```
832 \let\bbl@hyphlist\@empty
833 \let\bbl@hyphenation@\relax
834 \let\bbl@pttnlist\@empty
835 \let\bbl@patterns@\relax
836 \let\bbl@hymapsel=\@cclv
837 \def\bbl@patterns#1{%
838   \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
839       \csname l@#1\endcsname
840       \edef\bbl@tempa{#1}%
841     \else
842       \csname l@#1:\f@encoding\endcsname
843       \edef\bbl@tempa{#1:\f@encoding}%
```

```
844        \fi
845      \@expandtwoargs\bbl@usehooks{patterns}{{#1}{\bbl@tempa}}%
846      %  > luatex
847      \@ifundefined{bbl@hyphenation@}{}{% Can be \relax!
848        \begingroup
849          \bbl@xin@{,\number\language,}{,\bbl@hyphlist}%
850          \ifin@\else
851            \@expandtwoargs\bbl@usehooks{hyphenation}{{#1}{\bbl@tempa}}%
852            \hyphenation{%
853              \bbl@hyphenation@
854              \@ifundefined{bbl@hyphenation@#1}%
855                \@empty
856                {\space\csname bbl@hyphenation@#1\endcsname}}%
857            \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
858          \fi
859        \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*.

```
860 \def\hyphenrules#1{%
861   \edef\bbl@tempf{#1}%
862   \bbl@fixname\bbl@tempf
863   \bbl@iflanguage\bbl@tempf{%
864     \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
865     \ifx\languageshorthands\@undefined\else
866       \languageshorthands{none}%
867     \fi
868     \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax
869       \set@hyphenmins\tw@\thr@@\relax
870     \else
871       \expandafter\expandafter\expandafter\set@hyphenmins
872       \csname\bbl@tempf hyphenmins\endcsname\relax
873     \fi}}
874 \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.

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

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

```
879 \def\set@hyphenmins#1#2{%
880   \lefthyphenmin#1\relax
881   \righthyphenmin#2\relax}
```

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

```
882 \ifx\ProvidesFile\@undefined
883   \def\ProvidesLanguage#1[#2 #3 #4]{%
884     \wlog{Language: #1 #4 #3 <#2>}%
```

```
885       }
886 \else
887   \def\ProvidesLanguage#1{%
888     \begingroup
889       \catcode`\ 10 %
890       \@makeother\/%
891       \@ifnextchar[%
892         {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}}
893   \def\@provideslanguage#1[#2]{%
894     \wlog{Language: #1 #2}%
895     \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
896     \endgroup}
897 \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.

```
898 \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.

```
899 \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':

```
900 \providecommand\setlocale{%
901   \bbl@error
902     {Not yet available}%
903     {Find an armchair, sit down and wait}}
904 \let\uselocale\setlocale
905 \let\locale\setlocale
906 \let\selectlocale\setlocale
907 \let\localename\setlocale
908 \let\textlocale\setlocale
909 \let\textlanguage\setlocale
910 \let\languagetext\setlocale
```

8.2   Errors

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

\@noopterr   When the package was loaded without options not everything will work as expected. An error
                message is issued in that case.
                When the format knows about \PackageError it must be LaTeX 2_ε, so we can safely use its error
                handling interface. Otherwise we'll have to 'keep it simple'.
                Infos are not written to the console, but on the other hand many people think warnings are errors, so
                a further message type is defined: an important info which is sent to the console.

```
911 \edef\bbl@nulllanguage{\string\language=0}
912 \def\bbl@nocaption{\protect\bbl@nocaption@i}
913 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
914   \global\@namedef{#2}{\textbf{?#1?}}%
915   \@nameuse{#2}%
916   \edef\bbl@tempa{#1}%
917   \bbl@sreplace\bbl@tempa{name}{}%
918   \bbl@warning{% TODO.
919     \@backslashchar#1 not set for '\languagename'. Please,\\%
920     define it after the language has been loaded\\%
```

```
921     (typically in the preamble) with:\\%
922     \string\setlocalecaption{\languagename}{\bbl@tempa}{..}\\%
923     Reported}}
924 \def\bbl@tentative{\protect\bbl@tentative@i}
925 \def\bbl@tentative@i#1{%
926   \bbl@warning{%
927     Some functions for '#1' are tentative.\\%
928     They might not work as expected and their behavior\\%
929     could change in the future.\\%
930     Reported}}
931 \def\@nolanerr#1{%
932   \bbl@error
933     {You haven't defined the language '#1' yet.\\%
934      Perhaps you misspelled it or your installation\\%
935      is not complete}%
936     {Your command will be ignored, type <return> to proceed}}
937 \def\@nopatterns#1{%
938   \bbl@warning
939     {No hyphenation patterns were preloaded for\\%
940      the language '#1' into the format.\\%
941      Please, configure your TeX system to add them and\\%
942      rebuild the format. Now I will use the patterns\\%
943      preloaded for \bbl@nulllanguage\space instead}}
944 \let\bbl@usehooks\@gobbletwo
945 \ifx\bbl@onlyswitch\@empty\endinput\fi
946   % Here ended switch.def
```

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

```
947 \ifx\directlua\@undefined\else
948   \ifx\bbl@luapatterns\@undefined
949     \input luababel.def
950   \fi
951 \fi
952 ⟨⟨Basic macros⟩⟩
953 \bbl@trace{Compatibility with language.def}
954 \ifx\bbl@languages\@undefined
955   \ifx\directlua\@undefined
956     \openin1 = language.def % TODO. Remove hardcoded number
957     \ifeof1
958       \closein1
959       \message{I couldn't find the file language.def}
960     \else
961       \closein1
962       \begingroup
963         \def\addlanguage#1#2#3#4#5{%
964           \expandafter\ifx\csname lang@#1\endcsname\relax\else
965             \global\expandafter\let\csname l@#1\expandafter\endcsname
966               \csname lang@#1\endcsname
967           \fi}%
968         \def\uselanguage#1{}%
969         \input language.def
970       \endgroup
971     \fi
972   \fi
973   \chardef\l@english\z@
974 \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.

```
975 \def\addto#1#2{%
976   \ifx#1\@undefined
977     \def#1{#2}%
978   \else
979     \ifx#1\relax
980       \def#1{#2}%
981     \else
982       {\toks@\expandafter{#1#2}%
983         \xdef#1{\the\toks@}}%
984     \fi
985   \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?

```
986 \def\bbl@withactive#1#2{%
987   \begingroup
988     \lccode`~=`#2\relax
989     \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.

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

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

```
995 \def\bbl@redefine@long#1{%
996   \edef\bbl@tempa{\bbl@stripslash#1}%
997   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
998   \expandafter\long\expandafter\def\csname\bbl@tempa\endcsname}
999 \@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␣.

```
1000 \def\bbl@redefinerobust#1{%
1001   \edef\bbl@tempa{\bbl@stripslash#1}%
1002   \bbl@ifunset{\bbl@tempa\space}%
1003     {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
1004       \bbl@exp{\def\\#1{\\\protect\<\bbl@tempa\space>}}}%
1005     {\bbl@exp{\let\<org@\bbl@tempa>\<\bbl@tempa\space>}}%
1006   \@namedef{\bbl@tempa\space}}
1007 \@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.

```
1008 \bbl@trace{Hooks}
1009 \newcommand\AddBabelHook[3][]{%
1010   \bbl@ifunset{bbl@hk@#2}{\EnableBabelHook{#2}}{}%
1011   \def\bbl@tempa##1,#3=##2,##3\@empty{\def\bbl@tempb{##2}}%
1012   \expandafter\bbl@tempa\bbl@evargs,#3=,\@empty
1013   \bbl@ifunset{bbl@ev@#2@#3@#1}%
1014     {\bbl@csarg\bbl@add{ev@#3@#1}{\bbl@elth{#2}}}%
1015     {\bbl@csarg\let{ev@#2@#3@#1}\relax}%
1016   \bbl@csarg\newcommand{ev@#2@#3@#1}[\bbl@tempb]}
1017 \newcommand\EnableBabelHook[1]{\bbl@csarg\let{hk@#1}\@firstofone}
1018 \newcommand\DisableBabelHook[1]{\bbl@csarg\let{hk@#1}\@gobble}
1019 \def\bbl@usehooks#1#2{%
1020   \ifx\UseHook\@undefined\else\UseHook{babel/*/#1}\fi
1021   \def\bbl@elth##1{%
1022     \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#1@}#2}}%
1023   \bbl@cs{ev@#1@}%
1024   \ifx\languagename\@undefined\else % Test required for Plain (?)
1025     \ifx\UseHook\@undefined\else\UseHook{babel/\languagename/#1}\fi
1026     \def\bbl@elth##1{%
1027       \bbl@cs{hk@##1}{\bbl@cl{ev@##1@#1}#2}}%
1028     \bbl@cl{ev@#1}%
1029   \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).

```
1030 \def\bbl@evargs{,% <- don't delete this comma
1031   everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
1032   adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
1033   beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
1034   hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%
1035   beforestart=0,languagename=2}
1036 \ifx\NewHook\@undefined\else
1037   \def\bbl@tempa#1=#2\@@{\NewHook{babel/#1}}
1038   \bbl@foreach\bbl@evargs{\bbl@tempa#1\@@}
1039 \fi
```

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

```
1040 \bbl@trace{Defining babelensure}
1041 \newcommand\babelensure[2][]{%  TODO - revise test files
1042   \AddBabelHook{babel-ensure}{afterextras}{%
1043     \ifcase\bbl@select@type
1044       \bbl@cl{e}%
1045     \fi}%
1046   \begingroup
1047     \let\bbl@ens@include\@empty
1048     \let\bbl@ens@exclude\@empty
1049     \def\bbl@ens@fontenc{\relax}%
1050     \def\bbl@tempb##1{%
1051       \ifx\@empty##1\else\noexpand##1\expandafter\bbl@tempb\fi}%
1052     \edef\bbl@tempa{\bbl@tempb#1\@empty}%
```

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

```
1104 \bbl@trace{Macros for setting language files up}
1105 \def\bbl@ldfinit{%
1106   \let\bbl@screset\@empty
1107   \let\BabelStrings\bbl@opt@string
1108   \let\BabelOptions\@empty
1109   \let\BabelLanguages\relax
1110   \ifx\originalTeX\@undefined
1111     \let\originalTeX\@empty
1112   \else
1113     \originalTeX
1114   \fi}
1115 \def\LdfInit#1#2{%
1116   \chardef\atcatcode=\catcode`\@
1117   \catcode`\@=11\relax
1118   \chardef\eqcatcode=\catcode`\=
1119   \catcode`\==12\relax
1120   \expandafter\if\expandafter\@backslashchar
1121                 \expandafter\@car\string#2\@nil
1122     \ifx#2\@undefined\else
1123       \ldf@quit{#1}%
1124     \fi
1125   \else
1126     \expandafter\ifx\csname#2\endcsname\relax\else
1127       \ldf@quit{#1}%
1128     \fi
1129   \fi
1130   \bbl@ldfinit}
```

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

```
1131 \def\ldf@quit#1{%
1132   \expandafter\main@language\expandafter{#1}%
1133   \catcode`\@=\atcatcode \let\atcatcode\relax
1134   \catcode`\==\eqcatcode \let\eqcatcode\relax
1135   \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.

```
1136 \def\bbl@afterldf#1{% TODO. Merge into the next macro? Unused elsewhere
1137   \bbl@afterlang
1138   \let\bbl@afterlang\relax
1139   \let\BabelModifiers\relax
1140   \let\bbl@screset\relax}%
```

89

```
1141 \def\ldf@finish#1{%
1142   \loadlocalcfg{#1}%
1143   \bbl@afterldf{#1}%
1144   \expandafter\main@language\expandafter{#1}%
1145   \catcode`\@=\atcatcode \let\atcatcode\relax
1146   \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.

```
1147 \@onlypreamble\LdfInit
1148 \@onlypreamble\ldf@quit
1149 \@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.

```
1150 \def\main@language#1{%
1151   \def\bbl@main@language{#1}%
1152   \let\languagename\bbl@main@language % TODO. Set localename
1153   \bbl@id@assign
1154   \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.

```
1155 \def\bbl@beforestart{%
1156   \def\@nolanerr##1{%
1157     \bbl@warning{Undefined language '##1' in aux.\\Reported}}%
1158   \bbl@usehooks{beforestart}{}%
1159   \global\let\bbl@beforestart\relax}
1160 \AtBeginDocument{%
1161   {\@nameuse{bbl@beforestart}}%  Group!
1162   \if@filesw
1163     \providecommand\babel@aux[2]{}%
1164     \immediate\write\@mainaux{%
1165       \string\providecommand\string\babel@aux[2]{}}%
1166     \immediate\write\@mainaux{\string\@nameuse{bbl@beforestart}}%
1167   \fi
1168   \expandafter\selectlanguage\expandafter{\bbl@main@language}%
1169   \ifbbl@single  % must go after the line above.
1170     \renewcommand\selectlanguage[1]{}%
1171     \renewcommand\foreignlanguage[2]{#2}%
1172     \global\let\babel@aux\@gobbletwo  % Also as flag
1173   \fi
1174   \ifcase\bbl@engine\or\pagedir\bodydir\fi} % TODO - a better place
```

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

```
1175 \def\select@language@x#1{%
1176   \ifcase\bbl@select@type
1177     \bbl@ifsamestring\languagename{#1}{}{\select@language{#1}}%
1178   \else
1179     \select@language{#1}%
1180   \fi}
```

8.5 Shorthands

\bbl@add@special The macro \bbl@add@special is used to add a new character (or single character control sequence) to the macro \dospecials (and \@sanitize if LaTeX is used). It is used only at one place, namely when \initiate@active@char is called (which is ignored if the char has been made active before). Because \@sanitize can be undefined, we put the definition inside a conditional.

Items are added to the lists without checking its existence or the original catcode. It does not hurt, but should be fixed. It's already done with \nfss@catcodes, added in 3.10.

```
1181 \bbl@trace{Shorhands}
1182 \def\bbl@add@special#1{% 1:a macro like \", \?, etc.
1183   \bbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
1184   \bbl@ifunset{@sanitize}{}{\bbl@add\@sanitize{\@makeother#1}}%
1185   \ifx\nfss@catcodes\@undefined\else % TODO - same for above
1186     \begingroup
1187       \catcode`#1\active
1188       \nfss@catcodes
1189       \ifnum\catcode`#1=\active
1190         \endgroup
1191         \bbl@add\nfss@catcodes{\@makeother#1}%
1192       \else
1193         \endgroup
1194       \fi
1195   \fi}
```

\bbl@remove@special   The companion of the former macro is \bbl@remove@special. It removes a character from the set macros \dospecials and \@sanitize, but it is not used at all in the babel core.

```
1196 \def\bbl@remove@special#1{%
1197   \begingroup
1198     \def\x##1##2{\ifnum`#1=`##2\noexpand\@empty
1199                  \else\noexpand##1\noexpand##2\fi}%
1200     \def\do{\x\do}%
1201     \def\@makeother{\x\@makeother}%
1202   \edef\x{\endgroup
1203     \def\noexpand\dospecials{\dospecials}%
1204     \expandafter\ifx\csname @sanitize\endcsname\relax\else
1205       \def\noexpand\@sanitize{\@sanitize}%
1206     \fi}%
1207   \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).

```
1208 \def\bbl@active@def#1#2#3#4{%
1209   \@namedef{#3#1}{%
1210     \expandafter\ifx\csname#2@sh@#1@\endcsname\relax
1211       \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
1212     \else
1213       \bbl@afterfi\csname#2@sh@#1@\endcsname
1214     \fi}%
```

91

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.

```
1215  \long\@namedef{#3@arg#1}##1{%
1216    \expandafter\ifx\csname#2@sh@#1@\string##1@\endcsname\relax
1217      \bbl@afterelse\csname#4#1\endcsname##1%
1218    \else
1219      \bbl@afterfi\csname#2@sh@#1@\string##1@\endcsname
1220    \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.

```
1221 \def\initiate@active@char#1{%
1222   \bbl@ifunset{active@char\string#1}%
1223     {\bbl@withactive
1224       {\expandafter\@initiate@active@char\expandafter}#1\string#1#1}%
1225     {}}
```

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).

```
1226 \def\@initiate@active@char#1#2#3{%
1227   \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
1228   \ifx#1\@undefined
1229     \bbl@csarg\def{oridef@#2}{\def#1{\active@prefix#1\@undefined}}%
1230   \else
1231     \bbl@csarg\let{oridef@@#2}#1%
1232     \bbl@csarg\edef{oridef@#2}{%
1233       \let\noexpand#1%
1234       \expandafter\noexpand\csname bbl@oridef@@#2\endcsname}%
1235   \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*).

```
1236   \ifx#1#3\relax
1237     \expandafter\let\csname normal@char#2\endcsname#3%
1238   \else
1239     \bbl@info{Making #2 an active character}%
1240     \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
1241       \@namedef{normal@char#2}{%
1242         \textormath{#3}{\csname bbl@oridef@@#2\endcsname}}%
1243     \else
1244       \@namedef{normal@char#2}{#3}%
1245     \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).

```
1246     \bbl@restoreactive{#2}%
1247     \AtBeginDocument{%
1248       \catcode`#2\active
1249       \if@filesw
1250         \immediate\write\@mainaux{\catcode`\string#2\active}%
```

```
1251        \fi}%
1252     \expandafter\bbl@add@special\csname#2\endcsname
1253     \catcode`#2\active
1254   \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*⟩).

```
1255   \let\bbl@tempa\@firstoftwo
1256   \if\string^#2%
1257     \def\bbl@tempa{\noexpand\textormath}%
1258   \else
1259     \ifx\bbl@mathnormal\@undefined\else
1260       \let\bbl@tempa\bbl@mathnormal
1261     \fi
1262   \fi
1263   \expandafter\edef\csname active@char#2\endcsname{%
1264     \bbl@tempa
1265       {\noexpand\if@safe@actives
1266          \noexpand\expandafter
1267          \expandafter\noexpand\csname normal@char#2\endcsname
1268        \noexpand\else
1269          \noexpand\expandafter
1270          \expandafter\noexpand\csname bbl@doactive#2\endcsname
1271        \noexpand\fi}%
1272       {\expandafter\noexpand\csname normal@char#2\endcsname}}%
1273   \bbl@csarg\edef{doactive#2}{%
1274     \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

$$\verb|\active@prefix| \langle char \rangle \verb|\normal@char|\langle char \rangle$$

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

```
1275   \bbl@csarg\edef{active@#2}{%
1276     \noexpand\active@prefix\noexpand#1%
1277     \expandafter\noexpand\csname active@char#2\endcsname}%
1278   \bbl@csarg\edef{normal@#2}{%
1279     \noexpand\active@prefix\noexpand#1%
1280     \expandafter\noexpand\csname normal@char#2\endcsname}%
1281   \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.

```
1282   \bbl@active@def#2\user@group{user@active}{language@active}%
1283   \bbl@active@def#2\language@group{language@active}{system@active}%
1284   \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.

```
1285   \expandafter\edef\csname\user@group @sh@#2@@\endcsname
1286     {\expandafter\noexpand\csname normal@char#2\endcsname}%
1287   \expandafter\edef\csname\user@group @sh@#2@\string\protect@\endcsname
1288     {\expandafter\noexpand\csname user@active#2\endcsname}%
```

93

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.

```
1289    \if\string'#2%
1290      \let\prim@s\bbl@prim@s
1291      \let\active@math@prime#1%
1292    \fi
1293    \bbl@usehooks{initiateactive}{{#1}{#2}{#3}}}
```

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

```
1294 ⟨⟨∗More package options⟩⟩ ≡
1295 \DeclareOption{math=active}{}
1296 \DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
1297 ⟨⟨/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.

```
1298 \@ifpackagewith{babel}{KeepShorthandsActive}%
1299    {\let\bbl@restoreactive\@gobble}%
1300    {\def\bbl@restoreactive#1{%
1301      \bbl@exp{%
1302        \\\AfterBabelLanguage\\\CurrentOption
1303          {\catcode`#1=\the\catcode`#1\relax}%
1304        \\\AtEndOfPackage
1305          {\catcode`#1=\the\catcode`#1\relax}}}%
1306    \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.

```
1307 \def\bbl@sh@select#1#2{%
1308    \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
1309      \bbl@afterelse\bbl@scndcs
1310    \else
1311      \bbl@afterfi\csname#1@sh@#2@sel\endcsname
1312    \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.

```
1313 \begingroup
1314 \bbl@ifunset{ifincsname}% TODO. Ugly. Correct? Only Plain?
1315    {\gdef\active@prefix#1{%
1316      \ifx\protect\@typeset@protect
1317      \else
1318        \ifx\protect\@unexpandable@protect
1319          \noexpand#1%
1320        \else
1321          \protect#1%
1322        \fi
```

94

```
1323        \expandafter\@gobble
1324      \fi}}
1325    {\gdef\active@prefix#1{%
1326      \ifincsname
1327        \string#1%
1328        \expandafter\@gobble
1329      \else
1330        \ifx\protect\@typeset@protect
1331        \else
1332          \ifx\protect\@unexpandable@protect
1333            \noexpand#1%
1334          \else
1335            \protect#1%
1336          \fi
1337          \expandafter\expandafter\expandafter\@gobble
1338        \fi
1339      \fi}}
1340  \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*⟩.

```
1341  \newif\if@safe@actives
1342  \@safe@activesfalse
```

\bbl@restore@actives When the output routine kicks in while the active characters were made "safe" this must be undone in the headers to prevent unexpected typeset results. For this situation we define a command to make them "unsafe" again.

```
1343  \def\bbl@restore@actives{\if@safe@actives\@safe@activesfalse\fi}
```

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

```
1344  \chardef\bbl@activated\z@
1345  \def\bbl@activate#1{%
1346    \chardef\bbl@activated\@ne
1347    \bbl@withactive{\expandafter\let\expandafter}#1%
1348      \csname bbl@active@\string#1\endcsname}
1349  \def\bbl@deactivate#1{%
1350    \chardef\bbl@activated\tw@
1351    \bbl@withactive{\expandafter\let\expandafter}#1%
1352      \csname bbl@normal@\string#1\endcsname}
```

\bbl@firstcs These macros are used only as a trick when declaring shorthands.
\bbl@scndcs
```
1353  \def\bbl@firstcs#1#2{\csname#1\endcsname}
1354  \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.

95

```
1355 \def\babel@texpdf#1#2#3#4{%
1356   \ifx\texorpdfstring\@undefined
1357     \textormath{#1}{#3}%
1358   \else
1359     \texorpdfstring{\textormath{#1}{#3}}{#2}%
1360     % \texorpdfstring{\textormath{#1}{#3}}{\textormath{#2}{#4}}%
1361   \fi}
1362 %
1363 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
1364 \def\@decl@short#1#2#3\@nil#4{%
1365   \def\bbl@tempa{#3}%
1366   \ifx\bbl@tempa\@empty
1367     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
1368     \bbl@ifunset{#1@sh@\string#2@}{}%
1369       {\def\bbl@tempa{#4}%
1370        \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbl@tempa
1371        \else
1372          \bbl@info
1373            {Redefining #1 shorthand \string#2\\%
1374             in language \CurrentOption}%
1375        \fi}%
1376     \@namedef{#1@sh@\string#2@}{#4}%
1377   \else
1378     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs
1379     \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%
1380       {\def\bbl@tempa{#4}%
1381        \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
1382        \else
1383          \bbl@info
1384            {Redefining #1 shorthand \string#2\string#3\\%
1385             in language \CurrentOption}%
1386        \fi}%
1387     \@namedef{#1@sh@\string#2@\string#3@}{#4}%
1388   \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.

```
1389 \def\textormath{%
1390   \ifmmode
1391     \expandafter\@secondoftwo
1392   \else
1393     \expandafter\@firstoftwo
1394   \fi}
```

\user@group
\language@group
\system@group
The current concept of 'shorthands' supports three levels or groups of shorthands. For each level the name of the level or group is stored in a macro. The default is to have a user group; use language group 'english' and have a system group called 'system'.

```
1395 \def\user@group{user}
1396 \def\language@group{english} % TODO. I don't like defaults
1397 \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.

```
1398 \def\useshorthands{%
1399   \@ifstar\bbl@usesh@s{\bbl@usesh@x{}}}
1400 \def\bbl@usesh@s#1{%
1401   \bbl@usesh@x
```

```
1402    {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bbl@activate{#1}}}%
1403    {#1}}
1404 \def\bbl@usesh@x#1#2{%
1405   \bbl@ifshorthand{#2}%
1406     {\def\user@group{user}%
1407      \initiate@active@char{#2}%
1408      #1%
1409      \bbl@activate{#2}}%
1410     {\bbl@error
1411       {I can't declare a shorthand turned off (\string#2)}
1412       {Sorry, but you can't use shorthands which have been\\%
1413        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.

```
1414 \def\user@language@group{user@\language@group}
1415 \def\bbl@set@user@generic#1#2{%
1416   \bbl@ifunset{user@generic@active#1}%
1417     {\bbl@active@def#1\user@language@group{user@active}{user@generic@active}%
1418      \bbl@active@def#1\user@group{user@generic@active}{language@active}%
1419      \expandafter\edef\csname#2@sh@#1@@\endcsname{%
1420        \expandafter\noexpand\csname normal@char#1\endcsname}%
1421      \expandafter\edef\csname#2@sh@#1@\string\protect@\endcsname{%
1422        \expandafter\noexpand\csname user@active#1\endcsname}}%
1423   \@empty}
1424 \newcommand\defineshorthand[3][user]{%
1425   \edef\bbl@tempa{\zap@space#1 \@empty}%
1426   \bbl@for\bbl@tempb\bbl@tempa{%
1427     \if*\expandafter\@car\bbl@tempb\@nil
1428       \edef\bbl@tempb{user@\expandafter\@gobble\bbl@tempb}%
1429       \@expandtwoargs
1430         \bbl@set@user@generic{\expandafter\string\@car#2\@nil}\bbl@tempb
1431     \fi
1432     \declare@shorthand{\bbl@tempb}{#2}{#3}}}
```

\languageshorthands A user level command to change the language from which shorthands are used. Unfortunately, babel
                     currently does not keep track of defined groups, and therefore there is no way to catch a possible
                     change in casing to fix it in the same way languages names are fixed. [TODO].

```
1433 \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".

```
1434 \def\aliasshorthand#1#2{%
1435   \bbl@ifshorthand{#2}%
1436     {\expandafter\ifx\csname active@char\string#2\endcsname\relax
1437       \ifx\document\@notprerr
1438         \@notshorthand{#2}%
1439       \else
1440         \initiate@active@char{#2}%
1441         \expandafter\let\csname active@char\string#2\expandafter\endcsname
1442           \csname active@char\string#1\endcsname
1443         \expandafter\let\csname normal@char\string#2\expandafter\endcsname
1444           \csname normal@char\string#1\endcsname
1445         \bbl@activate{#2}%
```

97

```
1446        \fi
1447      \fi}%
1448    {\bbl@error
1449      {Cannot declare a shorthand turned off (\string#2)}
1450      {Sorry, but you cannot use shorthands which have been\\%
1451        turned off in the package options}}}
```

\@notshorthand

```
1452 \def\@notshorthand#1{%
1453   \bbl@error{%
1454     The character '\string #1' should be made a shorthand character;\\%
1455     add the command \string\useshorthands\string{#1\string} to
1456     the preamble.\\%
1457     I will ignore your instruction}%
1458   {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.

```
1459 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
1460 \DeclareRobustCommand*\shorthandoff{%
1461   \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
1462 \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.

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

```
1491        \bbl@afterfi\bbl@switch@sh#1%
1492    \fi}
```

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

```
1493 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
1494 \def\bbl@putsh#1{%
1495    \bbl@ifunset{bbl@active@\string#1}%
1496        {\bbl@putsh@i#1\@empty\@nnil}%
1497        {\csname bbl@active@\string#1\endcsname}}
1498 \def\bbl@putsh@i#1#2\@nnil{%
1499    \csname\language@group @sh@\string#1@%
1500        \ifx\@empty#2\else\string#2@\fi\endcsname}
1501 \ifx\bbl@opt@shorthands\@nnil\else
1502    \let\bbl@s@initiate@active@char\initiate@active@char
1503    \def\initiate@active@char#1{%
1504        \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
1505    \let\bbl@s@switch@sh\bbl@switch@sh
1506    \def\bbl@switch@sh#1#2{%
1507        \ifx#2\@nnil\else
1508            \bbl@afterfi
1509            \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
1510        \fi}
1511    \let\bbl@s@activate\bbl@activate
1512    \def\bbl@activate#1{%
1513        \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
1514    \let\bbl@s@deactivate\bbl@deactivate
1515    \def\bbl@deactivate#1{%
1516        \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
1517 \fi
```

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

```
1518 \newcommand\ifbabelshorthand[3]{\bbl@ifunset{bbl@active@\string#1}{#3}{#2}}
```

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

```
1519 \def\bbl@prim@s{%
1520    \prime\futurelet\@let@token\bbl@pr@m@s}
1521 \def\bbl@if@primes#1#2{%
1522    \ifx#1\@let@token
1523        \expandafter\@firstoftwo
1524    \else\ifx#2\@let@token
1525        \bbl@afterelse\expandafter\@firstoftwo
1526    \else
1527        \bbl@afterfi\expandafter\@secondoftwo
1528    \fi\fi}
1529 \begingroup
1530    \catcode`\^=7  \catcode`\*=\active  \lccode`\*=`\^
1531    \catcode`\'=12 \catcode`\"=\active  \lccode`\"=`\'
1532    \lowercase{%
1533    \gdef\bbl@pr@m@s{%
1534        \bbl@if@primes"'%
1535            \pr@@@s
1536            {\bbl@if@primes*^\pr@@@t\egroup}}}
1537 \endgroup
```

99

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).

```
1538 \initiate@active@char{~}
1539 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
1540 \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.

```
1541 \expandafter\def\csname OT1dqpos\endcsname{127}
1542 \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

```
1543 \ifx\f@encoding\@undefined
1544   \def\f@encoding{OT1}
1545 \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.

```
1546 \bbl@trace{Language attributes}
1547 \newcommand\languageattribute[2]{%
1548   \def\bbl@tempc{#1}%
1549   \bbl@fixname\bbl@tempc
1550   \bbl@iflanguage\bbl@tempc{%
1551     \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.

```
1552       \ifx\bbl@known@attribs\@undefined
1553         \in@false
1554       \else
1555         \bbl@xin@{,\bbl@tempc-##1,}{,\bbl@known@attribs,}%
1556       \fi
1557       \ifin@
1558         \bbl@warning{%
1559           You have more than once selected the attribute '##1'\\%
1560           for language #1. Reported}%
1561       \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.

```
1562         \bbl@exp{%
1563           \\\bbl@add@list\\\bbl@known@attribs{\bbl@tempc-##1}}%
1564         \edef\bbl@tempa{\bbl@tempc-##1}%
1565         \expandafter\bbl@ifknown@ttrib\expandafter{\bbl@tempa}\bbl@attributes%
1566         {\csname\bbl@tempc @attr@##1\endcsname}%
1567         {\@attrerr{\bbl@tempc}{##1}}%
1568     \fi}}}
1569 \@onlypreamble\languageattribute
```

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

```
1570 \newcommand*{\@attrerr}[2]{%
1571   \bbl@error
1572     {The attribute #2 is unknown for language #1.}%
1573     {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}.

```
1574 \def\bbl@declare@ttribute#1#2#3{%
1575   \bbl@xin@{,#2,}{,\BabelModifiers,}%
1576   \ifin@
1577     \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
1578   \fi
1579   \bbl@add@list\bbl@attributes{#1-#2}%
1580   \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.

```
1581 \def\bbl@ifattributeset#1#2#3#4{%
1582   \ifx\bbl@known@attribs\@undefined
1583     \in@false
1584   \else
1585     \bbl@xin@{,#1-#2,}{,\bbl@known@attribs,}%
1586   \fi
1587   \ifin@
1588     \bbl@afterelse#3%
1589   \else
1590     \bbl@afterfi#4%
1591   \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.

```
1592 \def\bbl@ifknown@ttrib#1#2{%
1593   \let\bbl@tempa\@secondoftwo
1594   \bbl@loopx\bbl@tempb{#2}{%
1595     \expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{,#1,}%
1596     \ifin@
1597       \let\bbl@tempa\@firstoftwo
1598     \else
1599     \fi}%
1600   \bbl@tempa}
```

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

```
1601 \def\bbl@clear@ttribs{%
1602   \ifx\bbl@attributes\@undefined\else
1603     \bbl@loopx\bbl@tempa{\bbl@attributes}{%
1604       \expandafter\bbl@clear@ttrib\bbl@tempa.
1605       }%
1606     \let\bbl@attributes\@undefined
```

```
1607    \fi}
1608 \def\bbl@clear@ttrib#1-#2.{%
1609    \expandafter\let\csname#1@attr@#2\endcsname\@undefined}
1610 \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
```
1611 \bbl@trace{Macros for saving definitions}
1612 \def\babel@beginsave{\babel@savecnt\z@}
```

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

```
1613 \newcount\babel@savecnt
1614 \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.

```
1615 \def\babel@save#1{%
1616    \expandafter\let\csname babel@\number\babel@savecnt\endcsname#1\relax
1617    \toks@\expandafter{\originalTeX\let#1=}%
1618    \bbl@exp{%
1619       \def\\\originalTeX{\the\toks@\<babel@\number\babel@savecnt>\relax}}%
1620    \advance\babel@savecnt\@ne}
1621 \def\babel@savevariable#1{%
1622    \toks@\expandafter{\originalTeX #1=}%
1623    \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.

```
1624 \def\bbl@frenchspacing{%
1625    \ifnum\the\sfcode`\.=\@m
1626       \let\bbl@nonfrenchspacing\relax
1627    \else
1628       \frenchspacing
1629       \let\bbl@nonfrenchspacing\nonfrenchspacing
1630    \fi}
1631 \let\bbl@nonfrenchspacing\nonfrenchspacing
1632 \let\bbl@elt\relax
1633 \edef\bbl@fs@chars{%
1634    \bbl@elt{\string.}\@m{3000}\bbl@elt{\string?}\@m{3000}%
1635    \bbl@elt{\string!}\@m{3000}\bbl@elt{\string:}\@m{2000}%
1636    \bbl@elt{\string;}\@m{1500}\bbl@elt{\string,}\@m{1250}}
1637 \def\bbl@pre@fs{%
```

---

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

```
1638    \def\bbl@elt##1##2##3{\sfcode`##1=\the\sfcode`##1\relax}%
1639    \edef\bbl@save@sfcodes{\bbl@fs@chars}}%
1640 \def\bbl@post@fs{%
1641    \bbl@save@sfcodes
1642    \edef\bbl@tempa{\bbl@cl{frspc}}%
1643    \edef\bbl@tempa{\expandafter\@car\bbl@tempa\@nil}%
1644    \if u\bbl@tempa          % do nothing
1645    \else\if n\bbl@tempa     % non french
1646      \def\bbl@elt##1##2##3{%
1647        \ifnum\sfcode`##1=##2\relax
1648          \babel@savevariable{\sfcode`##1}%
1649          \sfcode`##1=##3\relax
1650        \fi}%
1651      \bbl@fs@chars
1652    \else\if y\bbl@tempa      % french
1653      \def\bbl@elt##1##2##3{%
1654        \ifnum\sfcode`##1=##3\relax
1655          \babel@savevariable{\sfcode`##1}%
1656          \sfcode`##1=##2\relax
1657        \fi}%
1658      \bbl@fs@chars
1659    \fi\fi\fi}
```

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.

```
1660 \bbl@trace{Short tags}
1661 \def\babeltags#1{%
1662    \edef\bbl@tempa{\zap@space#1 \@empty}%
1663    \def\bbl@tempb##1=##2\@@{%
1664      \edef\bbl@tempc{%
1665        \noexpand\newcommand
1666        \expandafter\noexpand\csname ##1\endcsname{%
1667          \noexpand\protect
1668          \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}%
1669        \noexpand\newcommand
1670        \expandafter\noexpand\csname text##1\endcsname{%
1671          \noexpand\foreignlanguage{##2}}}%
1672      \bbl@tempc}%
1673    \bbl@for\bbl@tempa\bbl@tempa{%
1674      \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.

```
1675 \bbl@trace{Hyphens}
1676 \@onlypreamble\babelhyphenation
1677 \AtEndOfPackage{%
1678    \newcommand\babelhyphenation[2][\@empty]{%
1679      \ifx\bbl@hyphenation@\relax
1680        \let\bbl@hyphenation@\@empty
1681      \fi
1682      \ifx\bbl@hyphlist\@empty\else
1683        \bbl@warning{%
```

```
1684          You must not intermingle \string\selectlanguage\space and\\%
1685          \string\babelhyphenation\space or some exceptions will not\\%
1686          be taken into account. Reported}%
1687       \fi
1688       \ifx\@empty#1%
1689         \protected@edef\bbl@hyphenation@{\bbl@hyphenation@\space#2}%
1690       \else
1691         \bbl@vforeach{#1}{%
1692           \def\bbl@tempa{##1}%
1693           \bbl@fixname\bbl@tempa
1694           \bbl@iflanguage\bbl@tempa{%
1695             \bbl@csarg\protected@edef{hyphenation@\bbl@tempa}{%
1696               \bbl@ifunset{bbl@hyphenation@\bbl@tempa}%
1697                 {}%
1698                 {\csname bbl@hyphenation@\bbl@tempa\endcsname\space}%
1699             #2}}}%
1700       \fi}}
```

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

```
1701 \def\bbl@allowhyphens{\ifvmode\else\nobreak\hskip\z@skip\fi}
1702 \def\bbl@t@one{T1}
1703 \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.

```
1704 \newcommand\babelnullhyphen{\char\hyphenchar\font}
1705 \def\babelhyphen{\active@prefix\babelhyphen\bbl@hyphen}
1706 \def\bbl@hyphen{%
1707   \@ifstar{\bbl@hyphen@i @}{\bbl@hyphen@i\@empty}}
1708 \def\bbl@hyphen@i#1#2{%
1709   \bbl@ifunset{bbl@hy@#1#2\@empty}%
1710     {\csname bbl@#1usehyphen\endcsname{\discretionary{#2}{}{#2}}}%
1711     {\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.

```
1712 \def\bbl@usehyphen#1{%
1713   \leavevmode
1714   \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
1715   \nobreak\hskip\z@skip}
1716 \def\bbl@@usehyphen#1{%
1717   \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}
```

The following macro inserts the hyphen char.

```
1718 \def\bbl@hyphenchar{%
1719   \ifnum\hyphenchar\font=\m@ne
1720     \babelnullhyphen
1721   \else
1722     \char\hyphenchar\font
1723   \fi}
```

---

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

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.

```
1724 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}{}}}
1725 \def\bbl@hy@@soft{\bbl@@usehyphen{\discretionary{\bbl@hyphenchar}{}{}}}
1726 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
1727 \def\bbl@hy@@hard{\bbl@@usehyphen\bbl@hyphenchar}
1728 \def\bbl@hy@nobreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}}
1729 \def\bbl@hy@@nobreak{\mbox{\bbl@hyphenchar}}
1730 \def\bbl@hy@repeat{%
1731   \bbl@usehyphen{%
1732     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
1733 \def\bbl@hy@@repeat{%
1734   \bbl@@usehyphen{%
1735     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
1736 \def\bbl@hy@empty{\hskip\z@skip}
1737 \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.

```
1738 \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.

```
1739 \bbl@trace{Multiencoding strings}
1740 \def\bbl@toglobal#1{\global\let#1#1}
1741 \def\bbl@recatcode#1{% TODO. Used only once?
1742   \@tempcnta="7F
1743   \def\bbl@tempa{%
1744     \ifnum\@tempcnta>"FF\else
1745       \catcode\@tempcnta=#1\relax
1746       \advance\@tempcnta\@ne
1747       \expandafter\bbl@tempa
1748     \fi}%
1749   \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).

```
1750 \@ifpackagewith{babel}{nocase}%
1751   {\let\bbl@patchuclc\relax}%
1752   {\def\bbl@patchuclc{%
1753     \global\let\bbl@patchuclc\relax
1754     \g@addto@macro\@uclclist{\reserved@b{\reserved@b\bbl@uclc}}}%
```

```
1755     \gdef\bbl@uclc##1{%
1756       \let\bbl@encoded\bbl@encoded@uclc
1757       \bbl@ifunset{\languagename @bbl@uclc}% and resumes it
1758         {##1}%
1759         {\let\bbl@tempa##1\relax % Used by LANG@bbl@uclc
1760          \csname\languagename @bbl@uclc\endcsname}%
1761       {\bbl@tolower\@empty}{\bbl@toupper\@empty}}%
1762     \gdef\bbl@tolower{\csname\languagename @bbl@lc\endcsname}%
1763     \gdef\bbl@toupper{\csname\languagename @bbl@uc\endcsname}}}
```

```
1764 ⟨⟨∗More package options⟩⟩ ≡
1765 \DeclareOption{nocase}{}
1766 ⟨⟨/More package options⟩⟩
```

The following package options control the behavior of \SetString.

```
1767 ⟨⟨∗More package options⟩⟩ ≡
1768 \let\bbl@opt@strings\@nnil % accept strings=value
1769 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
1770 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
1771 \def\BabelStringsDefault{generic}
1772 ⟨⟨/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.

```
1773 \@onlypreamble\StartBabelCommands
1774 \def\StartBabelCommands{%
1775   \begingroup
1776   \bbl@recatcode{11}%
1777   ⟨⟨Macros local to BabelCommands⟩⟩
1778   \def\bbl@provstring##1##2{%
1779     \providecommand##1{##2}%
1780     \bbl@toglobal##1}%
1781   \global\let\bbl@scafter\@empty
1782   \let\StartBabelCommands\bbl@startcmds
1783   \ifx\BabelLanguages\relax
1784     \let\BabelLanguages\CurrentOption
1785   \fi
1786   \begingroup
1787   \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
1788   \StartBabelCommands}
1789 \def\bbl@startcmds{%
1790   \ifx\bbl@screset\@nnil\else
1791     \bbl@usehooks{stopcommands}{}%
1792   \fi
1793   \endgroup
1794   \begingroup
1795   \@ifstar
1796     {\ifx\bbl@opt@strings\@nnil
1797        \let\bbl@opt@strings\BabelStringsDefault
1798      \fi
1799      \bbl@startcmds@i}%
1800     \bbl@startcmds@i}
1801 \def\bbl@startcmds@i#1#2{%
1802   \edef\bbl@L{\zap@space#1 \@empty}%
1803   \edef\bbl@G{\zap@space#2 \@empty}%
1804   \bbl@startcmds@ii}
1805 \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.

```
1806 \newcommand\bbl@startcmds@ii[1][\@empty]{%
1807   \let\SetString\@gobbletwo
1808   \let\bbl@stringdef\@gobbletwo
1809   \let\AfterBabelCommands\@gobble
1810   \ifx\@empty#1%
1811     \def\bbl@sc@label{generic}%
1812     \def\bbl@encstring##1##2{%
1813       \ProvideTextCommandDefault##1{##2}%
1814       \bbl@toglobal##1%
1815       \expandafter\bbl@toglobal\csname\string?\string##1\endcsname}%
1816     \let\bbl@sctest\in@true
1817   \else
1818     \let\bbl@sc@charset\space % <- zapped below
1819     \let\bbl@sc@fontenc\space % <-    "        "
1820     \def\bbl@tempa##1=##2\@nil{%
1821       \bbl@csarg\edef{sc@\zap@space##1 \@empty}{##2 }}%
1822     \bbl@vforeach{label=#1}{\bbl@tempa##1\@nil}%
1823     \def\bbl@tempa##1 ##2{% space -> comma
1824       ##1%
1825       \ifx\@empty##2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa##2\fi}%
1826     \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%
1827     \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%
1828     \edef\bbl@sc@charset{\expandafter\zap@space\bbl@sc@charset\@empty}%
1829     \def\bbl@encstring##1##2{%
1830       \bbl@foreach\bbl@sc@fontenc{%
1831         \bbl@ifunset{T@####1}%
1832           {}%
1833           {\ProvideTextCommand##1{####1}{##2}%
1834            \bbl@toglobal##1%
1835            \expandafter
1836            \bbl@toglobal\csname####1\string##1\endcsname}}}%
1837     \def\bbl@sctest{%
1838       \bbl@xin@{,\bbl@opt@strings,}{,\bbl@sc@label,\bbl@sc@fontenc,}}%
1839   \fi
1840   \ifx\bbl@opt@strings\@nnil        % ie, no strings key -> defaults
1841   \else\ifx\bbl@opt@strings\relax   % ie, strings=encoded
1842     \let\AfterBabelCommands\bbl@aftercmds
1843     \let\SetString\bbl@setstring
1844     \let\bbl@stringdef\bbl@encstring
1845   \else       % ie, strings=value
1846     \bbl@sctest
1847     \ifin@
1848       \let\AfterBabelCommands\bbl@aftercmds
1849       \let\SetString\bbl@setstring
1850       \let\bbl@stringdef\bbl@provstring
1851     \fi\fi\fi
1852   \bbl@scswitch
1853   \ifx\bbl@G\@empty
1854     \def\SetString##1##2{%
```

107

```
1855        \bbl@error{Missing group for string \string##1}%
1856          {You must assign strings to some category, typically\\%
1857           captions or extras, but you set none}}%
1858    \fi
1859    \ifx\@empty#1%
1860      \bbl@usehooks{defaultcommands}{}%
1861    \else
1862      \@expandtwoargs
1863      \bbl@usehooks{encodedcommands}{{\bbl@sc@charset}{\bbl@sc@fontenc}}%
1864    \fi}
```

There are two versions of `\bbl@scswitch`. The first version is used when `ldf`s 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 `ldf`s), and the second one skips undefined languages (after babel has been loaded) .

```
1865 \def\bbl@forlang#1#2{%
1866    \bbl@for#1\bbl@L{%
1867      \bbl@xin@{,#1,}{,\BabelLanguages,}%
1868      \ifin@#2\relax\fi}}
1869 \def\bbl@scswitch{%
1870    \bbl@forlang\bbl@tempa{%
1871      \ifx\bbl@G\@empty\else
1872        \ifx\SetString\@gobbletwo\else
1873          \edef\bbl@GL{\bbl@G\bbl@tempa}%
1874          \bbl@xin@{,\bbl@GL,}{,\bbl@screset,}%
1875          \ifin@\else
1876            \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
1877            \xdef\bbl@screset{\bbl@screset,\bbl@GL}%
1878          \fi
1879        \fi
1880      \fi}}
1881 \AtEndOfPackage{%
1882    \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{}{#2}}}%
1883    \let\bbl@scswitch\relax}
1884 \@onlypreamble\EndBabelCommands
1885 \def\EndBabelCommands{%
1886    \bbl@usehooks{stopcommands}{}%
1887    \endgroup
1888    \endgroup
1889    \bbl@scafter}
1890 \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.

```
1891 \def\bbl@setstring#1#2{% eg, \prefacename{<string>}
1892    \bbl@forlang\bbl@tempa{%
1893      \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
1894      \bbl@ifunset{\bbl@LC}% eg, \germanchaptername
1895        {\bbl@exp{%
1896          \global\\\bbl@add\<\bbl@G\bbl@tempa>{\\\bbl@scset\\#1\<\bbl@LC>}}}%
```

```
1897        {}%
1898     \def\BabelString{#2}%
1899     \bbl@usehooks{stringprocess}{}%
1900     \expandafter\bbl@stringdef
1901        \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.

```
1902 \ifx\bbl@opt@strings\relax
1903   \def\bbl@scset#1#2{\def#1{\bbl@encoded#2}}
1904   \bbl@patchuclc
1905   \let\bbl@encoded\relax
1906   \def\bbl@encoded@uclc#1{%
1907     \@inmathwarn#1%
1908     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
1909       \expandafter\ifx\csname ?\string#1\endcsname\relax
1910         \TextSymbolUnavailable#1%
1911       \else
1912         \csname ?\string#1\endcsname
1913       \fi
1914     \else
1915       \csname\cf@encoding\string#1\endcsname
1916     \fi}
1917 \else
1918   \def\bbl@scset#1#2{\def#1{#2}}
1919 \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.

```
1920 ⟨⟨*Macros local to BabelCommands⟩⟩ ≡
1921 \def\SetStringLoop##1##2{%
1922     \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
1923     \count@\z@
1924     \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
1925       \advance\count@\@ne
1926       \toks@\expandafter{\bbl@tempa}%
1927       \bbl@exp{%
1928         \\\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
1929         \count@=\the\count@\relax}}}%
1930 ⟨⟨/Macros local to BabelCommands⟩⟩
```

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

```
1931 \def\bbl@aftercmds#1{%
1932   \toks@\expandafter{\bbl@scafter#1}%
1933   \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.

```
1934 ⟨⟨*Macros local to BabelCommands⟩⟩ ≡
1935   \newcommand\SetCase[3][]{%
1936     \bbl@patchuclc
1937     \bbl@forlang\bbl@tempa{%
1938       \expandafter\bbl@encstring
1939         \csname\bbl@tempa @bbl@uclc\endcsname{\bbl@tempa##1}%
1940       \expandafter\bbl@encstring
```

109

```
1941            \csname\bbl@tempa @bbl@uc\endcsname{##2}%
1942         \expandafter\bbl@encstring
1943            \csname\bbl@tempa @bbl@lc\endcsname{##3}}}%
1944 ⟨⟨/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.

```
1945 ⟨⟨*Macros local to BabelCommands⟩⟩ ≡
1946   \newcommand\SetHyphenMap[1]{%
1947     \bbl@forlang\bbl@tempa{%
1948       \expandafter\bbl@stringdef
1949         \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
1950 ⟨⟨/Macros local to BabelCommands⟩⟩
```

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

```
1951 \newcommand\BabelLower[2]{% one to one.
1952   \ifnum\lccode#1=#2\else
1953     \babel@savevariable{\lccode#1}%
1954     \lccode#1=#2\relax
1955   \fi}
1956 \newcommand\BabelLowerMM[4]{% many-to-many
1957   \@tempcnta=#1\relax
1958   \@tempcntb=#4\relax
1959   \def\bbl@tempa{%
1960     \ifnum\@tempcnta>#2\else
1961       \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
1962       \advance\@tempcnta#3\relax
1963       \advance\@tempcntb#3\relax
1964       \expandafter\bbl@tempa
1965     \fi}%
1966   \bbl@tempa}
1967 \newcommand\BabelLowerMO[4]{% many-to-one
1968   \@tempcnta=#1\relax
1969   \def\bbl@tempa{%
1970     \ifnum\@tempcnta>#2\else
1971       \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
1972       \advance\@tempcnta#3
1973       \expandafter\bbl@tempa
1974     \fi}%
1975   \bbl@tempa}
```

The following package options control the behavior of hyphenation mapping.

```
1976 ⟨⟨*More package options⟩⟩ ≡
1977 \DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\z@}
1978 \DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap\@ne}
1979 \DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\tw@}
1980 \DeclareOption{hyphenmap=other}{\chardef\bbl@opt@hyphenmap\thr@@}
1981 \DeclareOption{hyphenmap=other*}{\chardef\bbl@opt@hyphenmap4\relax}
1982 ⟨⟨/More package options⟩⟩
```

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

```
1983 \AtEndOfPackage{%
1984   \ifx\bbl@opt@hyphenmap\@undefined
1985     \bbl@xin@{,}{\bbl@language@opts}%
1986     \chardef\bbl@opt@hyphenmap\ifin@4\else\@ne\fi
1987   \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.

```
1988 \newcommand\setlocalecaption{%  TODO. Catch typos. What about ensure?
1989   \@ifstar\bbl@setcaption@s\bbl@setcaption@x}
1990 \def\bbl@setcaption@x#1#2#3{%  language caption-name string
1991   \bbl@trim@def\bbl@tempa{#2}%
1992   \bbl@xin@{.template}{\bbl@tempa}%
1993   \ifin@
1994     \bbl@ini@captions@template{#3}{#1}%
1995   \else
1996     \edef\bbl@tempd{%
1997       \expandafter\expandafter\expandafter
1998       \strip@prefix\expandafter\meaning\csname captions#1\endcsname}%
1999     \bbl@xin@
2000       {\expandafter\string\csname #2name\endcsname}%
2001       {\bbl@tempd}%
2002     \ifin@ % Renew caption
2003       \bbl@xin@{\string\bbl@scset}{\bbl@tempd}%
2004       \ifin@
2005         \bbl@exp{%
2006           \\\bbl@ifsamestring{\bbl@tempa}{\languagename}%
2007             {\\\bbl@scset\<#2name>\<#1#2name>}%
2008             {}}%
2009       \else % Old way converts to new way
2010         \bbl@ifunset{#1#2name}%
2011           {\bbl@exp{%
2012             \\\bbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}%
2013             \\\bbl@ifsamestring{\bbl@tempa}{\languagename}%
2014               {\def\<#2name>{\<#1#2name>}}%
2015               {}}}%
2016           {}%
2017       \fi
2018     \else
2019       \bbl@xin@{\string\bbl@scset}{\bbl@tempd}% New
2020       \ifin@ % New way
2021         \bbl@exp{%
2022           \\\bbl@add\<captions#1>{\\\bbl@scset\<#2name>\<#1#2name>}%
2023           \\\bbl@ifsamestring{\bbl@tempa}{\languagename}%
2024             {\\\bbl@scset\<#2name>\<#1#2name>}%
2025             {}}%
2026       \else  % Old way, but defined in the new way
2027         \bbl@exp{%
2028           \\\bbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}%
2029           \\\bbl@ifsamestring{\bbl@tempa}{\languagename}%
2030             {\def\<#2name>{\<#1#2name>}}%
2031             {}}%
2032       \fi%
2033     \fi
2034     \@namedef{#1#2name}{#3}%
2035     \toks@\expandafter{\bbl@captionslist}%
2036     \bbl@exp{\\\in@{\<#2name>}{\the\toks@}}%
2037     \ifin@\else
2038       \bbl@exp{\\\bbl@add\\\bbl@captionslist{\<#2name>}}%
2039       \bbl@toglobal\bbl@captionslist
2040     \fi
2041   \fi}
2042 % \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.

```
2043 \bbl@trace{Macros related to glyphs}
2044 \def\set@low@box#1{\setbox\tw@\hbox{,}\setbox\z@\hbox{#1}%
2045    \dimen\z@\ht\z@ \advance\dimen\z@ -\ht\tw@%
2046    \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.

```
2047 \def\save@sf@q#1{\leavevmode
2048  \begingroup
2049   \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
2050  \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.

```
2051 \ProvideTextCommand{\quotedblbase}{OT1}{%
2052  \save@sf@q{\set@low@box{\textquotedblright\/}%
2053    \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.

```
2054 \ProvideTextCommandDefault{\quotedblbase}{%
2055  \UseTextSymbol{OT1}{\quotedblbase}}
```

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

```
2056 \ProvideTextCommand{\quotesinglbase}{OT1}{%
2057  \save@sf@q{\set@low@box{\textquoteright\/}%
2058    \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.

```
2059 \ProvideTextCommandDefault{\quotesinglbase}{%
2060  \UseTextSymbol{OT1}{\quotesinglbase}}
```

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

```
2061 \ProvideTextCommand{\guillemetleft}{OT1}{%
2062  \ifmmode
2063   \ll
2064  \else
2065   \save@sf@q{\nobreak
2066     \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2067  \fi}
2068 \ProvideTextCommand{\guillemetright}{OT1}{%
2069  \ifmmode
2070   \gg
2071  \else
2072   \save@sf@q{\nobreak
2073     \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2074  \fi}
```

```
2075 \ProvideTextCommand{\guillemotleft}{OT1}{%
2076   \ifmmode
2077     \ll
2078   \else
2079     \save@sf@q{\nobreak
2080       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2081   \fi}
2082 \ProvideTextCommand{\guillemotright}{OT1}{%
2083   \ifmmode
2084     \gg
2085   \else
2086     \save@sf@q{\nobreak
2087       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2088   \fi}
```

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

```
2089 \ProvideTextCommandDefault{\guillemetleft}{%
2090   \UseTextSymbol{OT1}{\guillemetleft}}
2091 \ProvideTextCommandDefault{\guillemetright}{%
2092   \UseTextSymbol{OT1}{\guillemetright}}
2093 \ProvideTextCommandDefault{\guillemotleft}{%
2094   \UseTextSymbol{OT1}{\guillemotleft}}
2095 \ProvideTextCommandDefault{\guillemotright}{%
2096   \UseTextSymbol{OT1}{\guillemotright}}
```

\guilsinglleft The single guillemets are not available in OT1 encoding. They are faked.
\guilsinglright
```
2097 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2098   \ifmmode
2099     <%
2100   \else
2101     \save@sf@q{\nobreak
2102       \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
2103   \fi}
2104 \ProvideTextCommand{\guilsinglright}{OT1}{%
2105   \ifmmode
2106     >%
2107   \else
2108     \save@sf@q{\nobreak
2109       \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
2110   \fi}
```

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

```
2111 \ProvideTextCommandDefault{\guilsinglleft}{%
2112   \UseTextSymbol{OT1}{\guilsinglleft}}
2113 \ProvideTextCommandDefault{\guilsinglright}{%
2114   \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.

```
2115 \DeclareTextCommand{\ij}{OT1}{%
2116   i\kern-0.02em\bbl@allowhyphens j}
2117 \DeclareTextCommand{\IJ}{OT1}{%
2118   I\kern-0.02em\bbl@allowhyphens J}
2119 \DeclareTextCommand{\ij}{T1}{\char188}
2120 \DeclareTextCommand{\IJ}{T1}{\char156}
```

113

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

```
2121 \ProvideTextCommandDefault{\ij}{%
2122   \UseTextSymbol{OT1}{\ij}}
2123 \ProvideTextCommandDefault{\IJ}{%
2124   \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).

```
2125 \def\crrtic@{\hrule height0.1ex width0.3em}
2126 \def\crttic@{\hrule height0.1ex width0.33em}
2127 \def\ddj@{%
2128   \setbox0\hbox{d}\dimen@=\ht0
2129   \advance\dimen@1ex
2130   \dimen@.45\dimen@
2131   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2132   \advance\dimen@ii.5ex
2133   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2134 \def\DDJ@{%
2135   \setbox0\hbox{D}\dimen@=.55\ht0
2136   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2137   \advance\dimen@ii.15ex %                correction for the dash position
2138   \advance\dimen@ii-.15\fontdimen7\font %     correction for cmtt font
2139   \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
2140   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
2141 %
2142 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2143 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}
```

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

```
2144 \ProvideTextCommandDefault{\dj}{%
2145   \UseTextSymbol{OT1}{\dj}}
2146 \ProvideTextCommandDefault{\DJ}{%
2147   \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.

```
2148 \DeclareTextCommand{\SS}{OT1}{SS}
2149 \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
```
2150 \ProvideTextCommandDefault{\glq}{%
2151   \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}
```

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

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

```
2158        \textormath{\textquoteleft}{\mbox{\textquoteleft}}%
2159        \kern.07em\relax}}
2160 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}\grq}
```

  The 'german' double quotes.

```
2161 \ProvideTextCommandDefault{\glqq}{%
2162        \textormath{\quotedblbase}{\mbox{\quotedblbase}}}
```

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

```
2163 \ProvideTextCommand{\grqq}{T1}{%
2164        \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2165 \ProvideTextCommand{\grqq}{TU}{%
2166        \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2167 \ProvideTextCommand{\grqq}{OT1}{%
2168        \save@sf@q{\kern-.07em
2169            \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}%
2170            \kern.07em\relax}}
2171 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}
```

\flq \frq The 'french' single guillemets.

```
2172 \ProvideTextCommandDefault{\flq}{%
2173        \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
2174 \ProvideTextCommandDefault{\frq}{%
2175        \textormath{\guilsinglright}{\mbox{\guilsinglright}}}
```

\flqq \frqq The 'french' double guillemets.

```
2176 \ProvideTextCommandDefault{\flqq}{%
2177        \textormath{\guillemetleft}{\mbox{\guillemetleft}}}
2178 \ProvideTextCommandDefault{\frqq}{%
2179        \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 \umlautlow To be able to provide both positions of \" we provide two commands to switch the positioning, the default will be \umlauthigh (the normal positioning).

```
2180 \def\umlauthigh{%
2181        \def\bbl@umlauta##1{\leavevmode\bgroup%
2182            \expandafter\accent\csname\f@encoding dqpos\endcsname
2183            ##1\bbl@allowhyphens\egroup}%
2184        \let\bbl@umlaute\bbl@umlauta}
2185 \def\umlautlow{%
2186        \def\bbl@umlauta{\protect\lower@umlaut}}
2187 \def\umlautelow{%
2188        \def\bbl@umlaute{\protect\lower@umlaut}}
2189 \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.

```
2190 \expandafter\ifx\csname U@D\endcsname\relax
2191        \csname newdimen\endcsname\U@D
2192 \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.

```
2193 \def\lower@umlaut#1{%
2194   \leavevmode\bgroup
2195     \U@D 1ex%
2196     {\setbox\z@\hbox{%
2197       \expandafter\char\csname\f@encoding dqpos\endcsname}%
2198       \dimen@ -.45ex\advance\dimen@\ht\z@
2199       \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2200     \expandafter\accent\csname\f@encoding dqpos\endcsname
2201     \fontdimen5\font\U@D #1%
2202   \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).

```
2203 \AtBeginDocument{%
2204   \DeclareTextCompositeCommand{\"}{OT1}{a}{\bbl@umlauta{a}}%
2205   \DeclareTextCompositeCommand{\"}{OT1}{e}{\bbl@umlaute{e}}%
2206   \DeclareTextCompositeCommand{\"}{OT1}{i}{\bbl@umlaute{\i}}%
2207   \DeclareTextCompositeCommand{\"}{OT1}{\i}{\bbl@umlaute{\i}}%
2208   \DeclareTextCompositeCommand{\"}{OT1}{o}{\bbl@umlauta{o}}%
2209   \DeclareTextCompositeCommand{\"}{OT1}{u}{\bbl@umlauta{u}}%
2210   \DeclareTextCompositeCommand{\"}{OT1}{A}{\bbl@umlauta{A}}%
2211   \DeclareTextCompositeCommand{\"}{OT1}{E}{\bbl@umlaute{E}}%
2212   \DeclareTextCompositeCommand{\"}{OT1}{I}{\bbl@umlaute{I}}%
2213   \DeclareTextCompositeCommand{\"}{OT1}{O}{\bbl@umlauta{O}}%
2214   \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.

```
2215 \ifx\l@english\@undefined
2216   \chardef\l@english\z@
2217 \fi
2218 % The following is used to cancel rules in ini files (see Amharic).
2219 \ifx\l@unhyphenated\@undefined
2220   \newlanguage\l@unhyphenated
2221 \fi
```

8.13 Layout

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

```
2222 \bbl@trace{Bidi layout}
2223 \providecommand\IfBabelLayout[3]{#3}%
2224 \newcommand\BabelPatchSection[1]{%
2225   \@ifundefined{#1}{}{%
2226     \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
2227     \@namedef{#1}{%
2228       \@ifstar{\bbl@presec@s{#1}}%
2229               {\@dblarg{\bbl@presec@x{#1}}}}}}
```

```
2230 \def\bbl@presec@x#1[#2]#3{%
2231   \bbl@exp{%
2232     \\\select@language@x{\bbl@main@language}%
2233     \\\bbl@cs{sspre@#1}%
2234     \\\bbl@cs{ss@#1}%
2235       [\\\foreignlanguage{\languagename}{\unexpanded{#2}}]%
2236       {\\\foreignlanguage{\languagename}{\unexpanded{#3}}}%
2237     \\\select@language@x{\languagename}}}
2238 \def\bbl@presec@s#1#2{%
2239   \bbl@exp{%
2240     \\\select@language@x{\bbl@main@language}%
2241     \\\bbl@cs{sspre@#1}%
2242     \\\bbl@cs{ss@#1}*%
2243       {\\\foreignlanguage{\languagename}{\unexpanded{#2}}}%
2244     \\\select@language@x{\languagename}}}
2245 \IfBabelLayout{sectioning}%
2246   {\BabelPatchSection{part}%
2247    \BabelPatchSection{chapter}%
2248    \BabelPatchSection{section}%
2249    \BabelPatchSection{subsection}%
2250    \BabelPatchSection{subsubsection}%
2251    \BabelPatchSection{paragraph}%
2252    \BabelPatchSection{subparagraph}%
2253    \def\babel@toc#1{%
2254      \select@language@x{\bbl@main@language}}}{}
2255 \IfBabelLayout{captions}%
2256   {\BabelPatchSection{caption}}{}
```

8.14   Load engine specific macros

```
2257 \bbl@trace{Input engine specific macros}
2258 \ifcase\bbl@engine
2259   \input txtbabel.def
2260 \or
2261   \input luababel.def
2262 \or
2263   \input xebabel.def
2264 \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.

```
2265 \bbl@trace{Creating languages and reading ini files}
2266 \let\bbl@extend@ini\@gobble
2267 \newcommand\babelprovide[2][]{%
2268   \let\bbl@savelangname\languagename
2269   \edef\bbl@savelocaleid{\the\localeid}%
2270   % Set name and locale id
2271   \edef\languagename{#2}%
2272   \bbl@id@assign
2273   % Initialize keys
2274   \let\bbl@KVP@captions\@nil
2275   \let\bbl@KVP@date\@nil
2276   \let\bbl@KVP@import\@nil
2277   \let\bbl@KVP@main\@nil
2278   \let\bbl@KVP@script\@nil
2279   \let\bbl@KVP@language\@nil
```

117

```
2280    \let\bbl@KVP@hyphenrules\@nil
2281    \let\bbl@KVP@linebreaking\@nil
2282    \let\bbl@KVP@justification\@nil
2283    \let\bbl@KVP@mapfont\@nil
2284    \let\bbl@KVP@maparabic\@nil
2285    \let\bbl@KVP@mapdigits\@nil
2286    \let\bbl@KVP@intraspace\@nil
2287    \let\bbl@KVP@intrapenalty\@nil
2288    \let\bbl@KVP@onchar\@nil
2289    \let\bbl@KVP@transforms\@nil
2290    \global\let\bbl@release@transforms\@empty
2291    \let\bbl@KVP@alph\@nil
2292    \let\bbl@KVP@Alph\@nil
2293    \let\bbl@KVP@labels\@nil
2294    \bbl@csarg\let{KVP@labels*}\@nil
2295    \global\let\bbl@inidata\@empty
2296    \global\let\bbl@extend@ini\@gobble
2297    \gdef\bbl@key@list{;}%
2298    \bbl@forkv{#1}{%  TODO - error handling
2299      \in@{/}{##1}%
2300      \ifin@
2301        \global\let\bbl@extend@ini\bbl@extend@ini@aux
2302        \bbl@renewinikey##1\@@{##2}%
2303      \else
2304        \bbl@csarg\def{KVP@##1}{##2}%
2305      \fi}%
2306    \chardef\bbl@howloaded=% 0:none; 1:ldf without ini; 2:ini
2307      \bbl@ifunset{date#2}\z@{\bbl@ifunset{bbl@llevel@#2}\@ne\tw@}%
2308    % == init ==
2309    \ifx\bbl@screset\@undefined
2310      \bbl@ldfinit
2311    \fi
2312    % ==
2313    \let\bbl@lbkflag\relax % \@empty = do setup linebreak
2314    \ifcase\bbl@howloaded
2315      \let\bbl@lbkflag\@empty % new
2316    \else
2317      \ifx\bbl@KVP@hyphenrules\@nil\else
2318        \let\bbl@lbkflag\@empty
2319      \fi
2320      \ifx\bbl@KVP@import\@nil\else
2321        \let\bbl@lbkflag\@empty
2322      \fi
2323    \fi
2324    % == import, captions ==
2325    \ifx\bbl@KVP@import\@nil\else
2326      \bbl@exp{\\\bbl@ifblank{\bbl@KVP@import}}%
2327        {\ifx\bbl@initoload\relax
2328          \begingroup
2329            \def\BabelBeforeIni##1##2{\gdef\bbl@KVP@import{##1}\endinput}%
2330            \bbl@input@texini{#2}%
2331          \endgroup
2332        \else
2333          \xdef\bbl@KVP@import{\bbl@initoload}%
2334        \fi}%
2335        {}%
2336    \fi
2337    \ifx\bbl@KVP@captions\@nil
2338      \let\bbl@KVP@captions\bbl@KVP@import
```

```
2339    \fi
2340    % ==
2341    \ifx\bbl@KVP@transforms\@nil\else
2342      \bbl@replace\bbl@KVP@transforms{ }{,}%
2343    \fi
2344    % == Load ini ==
2345    \ifcase\bbl@howloaded
2346      \bbl@provide@new{#2}%
2347    \else
2348      \bbl@ifblank{#1}%
2349        {}%  With \bbl@load@basic below
2350        {\bbl@provide@renew{#2}}%
2351    \fi
2352    % Post tasks
2353    % ----------
2354    % == subsequent calls after the first provide for a locale ==
2355    \ifx\bbl@inidata\@empty\else
2356      \bbl@extend@ini{#2}%
2357    \fi
2358    % == ensure captions ==
2359    \ifx\bbl@KVP@captions\@nil\else
2360      \bbl@ifunset{bbl@extracaps@#2}%
2361        {\bbl@exp{\\\babelensure[exclude=\\\today]{#2}}}%
2362        {\bbl@exp{\\\babelensure[exclude=\\\today,
2363                 include=\[bbl@extracaps@#2]]{#2}}}%
2364      \bbl@ifunset{bbl@ensure@\languagename}%
2365        {\bbl@exp{%
2366          \\\DeclareRobustCommand\<bbl@ensure@\languagename>[1]{%
2367            \\\foreignlanguage{\languagename}%
2368            {####1}}}}%
2369        {}%
2370      \bbl@exp{%
2371        \\\bbl@toglobal\<bbl@ensure@\languagename>%
2372        \\\bbl@toglobal\<bbl@ensure@\languagename\space>}%
2373    \fi
2374    % ==
2375    % At this point all parameters are defined if 'import'. Now we
2376    % execute some code depending on them. But what about if nothing was
2377    % imported? We just set the basic parameters, but still loading the
2378    % whole ini file.
2379    \bbl@load@basic{#2}%
2380    % == script, language ==
2381    % Override the values from ini or defines them
2382    \ifx\bbl@KVP@script\@nil\else
2383      \bbl@csarg\edef{sname@#2}{\bbl@KVP@script}%
2384    \fi
2385    \ifx\bbl@KVP@language\@nil\else
2386      \bbl@csarg\edef{lname@#2}{\bbl@KVP@language}%
2387    \fi
2388     % == onchar ==
2389    \ifx\bbl@KVP@onchar\@nil\else
2390      \bbl@luahyphenate
2391      \directlua{
2392        if Babel.locale_mapped == nil then
2393          Babel.locale_mapped = true
2394          Babel.linebreaking.add_before(Babel.locale_map)
2395          Babel.loc_to_scr = {}
2396          Babel.chr_to_loc = Babel.chr_to_loc or {}
2397        end}%
```

```
2398      \bbl@xin@{ ids }{ \bbl@KVP@onchar\space}%
2399      \ifin@
2400        \ifx\bbl@starthyphens\@undefined % Needed if no explicit selection
2401          \AddBabelHook{babel-onchar}{beforestart}{{\bbl@starthyphens}}%
2402        \fi
2403        \bbl@exp{\\\bbl@add\\\bbl@starthyphens
2404          {\\\bbl@patterns@lua{\languagename}}}%
2405        % TODO - error/warning if no script
2406        \directlua{
2407          if Babel.script_blocks['\bbl@cl{sbcp}'] then
2408            Babel.loc_to_scr[\the\localeid] =
2409              Babel.script_blocks['\bbl@cl{sbcp}']
2410            Babel.locale_props[\the\localeid].lc = \the\localeid\space
2411            Babel.locale_props[\the\localeid].lg = \the\@nameuse{l@\languagename}\space
2412          end
2413        }%
2414      \fi
2415      \bbl@xin@{ fonts }{ \bbl@KVP@onchar\space}%
2416      \ifin@
2417        \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
2418        \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
2419        \directlua{
2420          if Babel.script_blocks['\bbl@cl{sbcp}'] then
2421            Babel.loc_to_scr[\the\localeid] =
2422              Babel.script_blocks['\bbl@cl{sbcp}']
2423          end}%
2424        \ifx\bbl@mapselect\@undefined  % TODO. almost the same as mapfont
2425          \AtBeginDocument{%
2426            \bbl@patchfont{{\bbl@mapselect}}%
2427            {\selectfont}}%
2428          \def\bbl@mapselect{%
2429            \let\bbl@mapselect\relax
2430            \edef\bbl@prefontid{\fontid\font}}%
2431          \def\bbl@mapdir##1{%
2432            {\def\languagename{##1}%
2433             \let\bbl@ifrestoring\@firstoftwo % To avoid font warning
2434             \bbl@switchfont
2435             \directlua{
2436               Babel.locale_props[\the\csname bbl@id@@##1\endcsname]%
2437                       ['/\bbl@prefontid'] = \fontid\font\space}}}%
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}{}%
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]
```

122

```
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 reload 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
```

124

```
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
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}
```

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
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
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 %
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
```

130

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\\\localecounter{#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    %
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" 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]==]
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
```

135

```
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%
```

136

```
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.}}
```

137

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
```

141

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

```
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}
```

148

```
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=\\%
3988       headfoot=, strings=, config=, hyphenmap=, or a language name.}}}
```

Now, we set a few language options whose names are different from ldf files. These declarations are preserved for backwards compatibility, but they must be eventually removed. Use proxy files instead.

```
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 existing languages (note this list also contains the language given with main as the last element). If not declared above, the names of the option and the file are the same. There are two steps – first process option names and collect the result, which then do the actual declarations.

To allow multiple overlapping replacements, commas in \bbl@language@opts are doubled.

```
4021 \let\bbl@elt\relax
4022 \let\bbl@tempe\@empty
4023 \bbl@foreach\@classoptionslist{%
4024   \bbl@xin@{,#1,$}{\bbl@language@opts$}% Match last
4025   \ifin@\else
4026     \bbl@xin@{,#1,}{\bbl@language@opts}% Match non-last
4027     \ifin@
4028       \bbl@replace\bbl@language@opts{,#1,}{,,,}%
4029       \edef\bbl@tempe{\bbl@tempe\bbl@elt{3}{#1}}%
4030     \else
4031       \babel@savecnt\z@ % Use as temp
4032       \ifnum\bbl@iniflag<\thr@@ % Optimization: 3 = always ini
4033         \IfFileExists{#1.ldf}{\advance\babel@savecnt\@ne}{}%
4034       \fi
4035       \ifnum\bbl@iniflag>\z@ % Optimization: 0 = always ldf
4036         \IfFileExists{babel-#1.tex}{\advance\babel@savecnt\tw@}{}%
4037       \fi
4038       \ifnum\babel@savecnt>\z@
4039         \edef\bbl@tempe{\bbl@tempe\bbl@elt{\the\babel@savecnt}{#1}}%
4040       \fi
4041     \fi
4042   \fi}
4043 %
4044 \let\bbl@savemain\@empty
4045 \bbl@foreach\bbl@language@opts{%
4046   \edef\bbl@tempe{\bbl@tempe\bbl@elt{3}{#1}}}
4047 \def\bbl@elt#1#2#3{%
4048   \ifx#3\relax % if last
4049     \bbl@ifunset{ds@#2}{}%
4050       {\bbl@exp{\def\\\bbl@savemain{\\\DeclareOption{#2}{\[ds@#2]}}}}%
4051     \bbl@add\bbl@savemain{\bbl@elt{#1}{#2}}% Save main
4052     \DeclareOption{#2}{}%
4053   \else
4054     \ifnum\bbl@iniflag<\tw@ % other as ldf
4055       \ifodd#1\relax % Class: if ldf exists 1,3. Package: always 3
4056         \bbl@ifunset{ds@#2}%
4057           {\DeclareOption{#2}{\bbl@load@language{#2}}}%
4058           {}%
4059       \fi
4060     \else % other as ini
4061       \ifnum#1>\@ne %  % Class: if ini exists 2,3. Package: always 3
4062         \DeclareOption{#2}{%
4063           \bbl@ldfinit
4064           \babelprovide[import]{#2}%
4065           \bbl@afterldf{}}%
4066       \fi
```

```
4067        \fi
4068     \fi
4069     #3}
4070 \bbl@tempe\relax % \relax catches last
```

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.

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

```
4071 \def\AfterBabelLanguage#1{%
4072     \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang}{}}
4073 \DeclareOption*{}
4074 \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.

```
4075 \bbl@trace{Option 'main'}
4076 \ifx\bbl@opt@main\@nnil
4077     \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}
4078     \let\bbl@tempc\@empty
4079     \bbl@for\bbl@tempb\bbl@tempa{%
4080         \bbl@xin@{,\bbl@tempb,}{,\bbl@loaded,}%
4081         \ifin@\edef\bbl@tempc{\bbl@tempb}\fi}
4082     \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
4083     \expandafter\bbl@tempa\bbl@loaded,\@nnil
4084     \ifx\bbl@tempb\bbl@tempc\else
4085         \bbl@warning{%
4086             Last declared language option is '\bbl@tempc',\\%
4087             but the last processed one was '\bbl@tempb'.\\%
4088             The main language can't be set as both a global\\%
4089             and a package option. Use 'main=\bbl@tempc' as\\%
4090             option. Reported}%
4091     \fi
4092 \fi
4093 \def\bbl@elt#1#2{% main
4094     \ifodd\bbl@iniflag % as ini = 1(=), 3(*=)
4095         \ifnum#1>\@ne %  % Class: if ini exists 2,3. Package: always 3
4096             \def\CurrentOption{#2}% Directly, because luatexbase
4097             \bbl@ldfinit
4098             \babelprovide[\bbl@opt@provide,main,import]{#2}%
4099             \bbl@afterldf{}%
4100             \DeclareOption{#2}{}%
4101         \fi
4102     \else % as ldf = 0(no), 2(+=)
4103         \ifodd#1\relax % Class: if ldf exists 1,3. Package: always 3
4104             \bbl@ifunset{ds@#2}%
4105                 {\DeclareOption{#2}{\bbl@load@language{#2}}}%
4106                 {}%
4107             \ExecuteOptions{#2}%
4108         \fi
4109     \fi}
4110 \bbl@savemain
4111 \DeclareOption*{}%
4112 \ProcessOptions*
```

```
4113 \def\AfterBabelLanguage{%
4114   \bbl@error
4115     {Too late for \string\AfterBabelLanguage}%
4116     {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.

```
4117 \ifx\bbl@main@language\@undefined
4118   \bbl@info{%
4119     You haven't specified a language. I'll use 'nil'\\%
4120     as the main language. Reported}
4121     \bbl@load@language{nil}
4122 \fi
4123 ⟨/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

```
4124 ⟨*kernel⟩
4125 \let\bbl@onlyswitch\@empty
4126 \input babel.def
4127 \let\bbl@onlyswitch\@undefined
4128 ⟨/kernel⟩
4129 ⟨*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.

```
4130 ⟨⟨Make sure ProvidesFile is defined⟩⟩
4131 \ProvidesFile{hyphen.cfg}[⟨⟨date⟩⟩ ⟨⟨version⟩⟩ Babel hyphens]
4132 \xdef\bbl@format{\jobname}
4133 \def\bbl@version{⟨⟨version⟩⟩}
4134 \def\bbl@date{⟨⟨date⟩⟩}
4135 \ifx\AtBeginDocument\@undefined
4136   \def\@empty{}
4137 \fi
4138 ⟨⟨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.

```
4139 \def\process@line#1#2 #3 #4 {%
4140   \ifx=#1%
```

155

```
4141        \process@synonym{#2}%
4142    \else
4143        \process@language{#1#2}{#3}{#4}%
4144    \fi
4145    \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.

```
4146 \toks@{}
4147 \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.

```
4148 \def\process@synonym#1{%
4149    \ifnum\last@language=\m@ne
4150        \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4151    \else
4152        \expandafter\chardef\csname l@#1\endcsname\last@language
4153        \wlog{\string\l@#1=\string\language\the\last@language}%
4154        \expandafter\let\csname #1hyphenmins\expandafter\endcsname
4155            \csname\languagename hyphenmins\endcsname
4156        \let\bbl@elt\relax
4157        \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}{}{}}%
4158    \fi}
```

\process@language   The macro \process@language is used to process a non-empty line from the 'configuration file'. It has three arguments, each delimited by white space. The first argument is the 'name' of a language; the second is the name of the file that contains the patterns. The optional third argument is the name of a file containing hyphenation exceptions.
The first thing to do is call \addlanguage to allocate a pattern register and to make that register 'active'. Then the pattern file is read.
For some hyphenation patterns it is needed to load them with a specific font encoding selected. This can be specified in the file language.dat by adding for instance ':T1' to the name of the language. The macro \bbl@get@enc extracts the font encoding from the language name and stores it in \bbl@hyph@enc. The latter can be used in hyphenation files if you need to set a behavior depending on the given encoding (it is set to empty if no encoding is given).
Pattern files may contain assignments to \lefthyphenmin and \righthyphenmin. TEX does not keep track of these assignments. Therefore we try to detect such assignments and store them in the \⟨lang⟩hyphenmins macro. When no assignments were made we provide a default setting.
Some pattern files contain changes to the \lccode en \uccode arrays. Such changes should remain local to the language; therefore we process the pattern file in a group; the \patterns command acts globally so its effect will be remembered.
Then we globally store the settings of \lefthyphenmin and \righthyphenmin and close the group.
When the hyphenation patterns have been processed we need to see if a file with hyphenation exceptions needs to be read. This is the case when the third argument is not empty and when it does not contain a space token. (Note however there is no need to save hyphenation exceptions into the format.)
\bbl@languages saves a snapshot of the loaded languages in the form
\bbl@elt{⟨language-name⟩}{⟨number⟩} {⟨patterns-file⟩}{⟨exceptions-file⟩}. Note the last 2 arguments are empty in 'dialects' defined in language.dat with =. Note also the language name can have encoding info.
Finally, if the counter \language is equal to zero we execute the synonyms stored.

```
4159 \def\process@language#1#2#3{%
4160    \expandafter\addlanguage\csname l@#1\endcsname
4161    \expandafter\language\csname l@#1\endcsname
4162    \edef\languagename{#1}%
```

```
4163    \bbl@hook@everylanguage{#1}%
4164    %  > luatex
4165    \bbl@get@enc#1::\@@@
4166    \begingroup
4167      \lefthyphenmin\m@ne
4168      \bbl@hook@loadpatterns{#2}%
4169      %  > luatex
4170      \ifnum\lefthyphenmin=\m@ne
4171      \else
4172        \expandafter\xdef\csname #1hyphenmins\endcsname{%
4173          \the\lefthyphenmin\the\righthyphenmin}%
4174      \fi
4175    \endgroup
4176    \def\bbl@tempa{#3}%
4177    \ifx\bbl@tempa\@empty\else
4178      \bbl@hook@loadexceptions{#3}%
4179      %  > luatex
4180    \fi
4181    \let\bbl@elt\relax
4182    \edef\bbl@languages{%
4183      \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
4184    \ifnum\the\language=\z@
4185      \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4186        \set@hyphenmins\tw@\thr@@\relax
4187      \else
4188        \expandafter\expandafter\expandafter\set@hyphenmins
4189          \csname #1hyphenmins\endcsname
4190      \fi
4191      \the\toks@
4192      \toks@{}%
4193    \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.

```
4194 \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.

```
4195 \def\bbl@hook@everylanguage#1{}
4196 \def\bbl@hook@loadpatterns#1{\input #1\relax}
4197 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
4198 \def\bbl@hook@loadkernel#1{%
4199   \def\addlanguage{\csname newlanguage\endcsname}%
4200   \def\adddialect##1##2{%
4201     \global\chardef##1##2\relax
4202     \wlog{\string##1 = a dialect from \string\language##2}}%
4203   \def\iflanguage##1{%
4204     \expandafter\ifx\csname l@##1\endcsname\relax
4205       \@nolanerr{##1}%
4206     \else
4207       \ifnum\csname l@##1\endcsname=\language
4208         \expandafter\expandafter\expandafter\@firstoftwo
4209       \else
4210         \expandafter\expandafter\expandafter\@secondoftwo
4211       \fi
4212     \fi}%
4213   \def\providehyphenmins##1##2{%
```

```
4214      \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
4215        \@namedef{##1hyphenmins}{##2}%
4216      \fi}%
4217    \def\set@hyphenmins##1##2{%
4218      \lefthyphenmin##1\relax
4219      \righthyphenmin##2\relax}%
4220    \def\selectlanguage{%
4221      \errhelp{Selecting a language requires a package supporting it}%
4222      \errmessage{Not loaded}}%
4223    \let\foreignlanguage\selectlanguage
4224    \let\otherlanguage\selectlanguage
4225    \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4226    \def\bbl@usehooks##1##2{}% TODO. Temporary!!
4227    \def\setlocale{%
4228      \errhelp{Find an armchair, sit down and wait}%
4229      \errmessage{Not yet available}}%
4230    \let\uselocale\setlocale
4231    \let\locale\setlocale
4232    \let\selectlocale\setlocale
4233    \let\localename\setlocale
4234    \let\textlocale\setlocale
4235    \let\textlanguage\setlocale
4236    \let\languagetext\setlocale}
4237 \begingroup
4238    \def\AddBabelHook#1#2{%
4239      \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
4240        \def\next{\toks1}%
4241      \else
4242        \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname####1}%
4243      \fi
4244      \next}
4245    \ifx\directlua\@undefined
4246      \ifx\XeTeXinputencoding\@undefined\else
4247        \input xebabel.def
4248      \fi
4249    \else
4250      \input luababel.def
4251    \fi
4252    \openin1 = babel-\bbl@format.cfg
4253    \ifeof1
4254    \else
4255      \input babel-\bbl@format.cfg\relax
4256    \fi
4257    \closein1
4258 \endgroup
4259 \bbl@hook@loadkernel{switch.def}
```

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

```
4260 \openin1 = language.dat
```

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

```
4261 \def\languagename{english}%
4262 \ifeof1
4263   \message{I couldn't find the file language.dat,\space
4264           I will try the file hyphen.tex}
4265   \input hyphen.tex\relax
4266   \chardef\l@english\z@
4267 \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.

```
4268    \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.

```
4269    \loop
4270      \endlinechar\m@ne
4271      \read1 to \bbl@line
4272      \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.

```
4273      \if T\ifeof1F\fi T\relax
4274        \ifx\bbl@line\@empty\else
4275          \edef\bbl@line{\bbl@line\space\space\space}%
4276          \expandafter\process@line\bbl@line\relax
4277        \fi
4278    \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.

```
4279    \begingroup
4280      \def\bbl@elt#1#2#3#4{%
4281        \global\language=#2\relax
4282        \gdef\languagename{#1}%
4283        \def\bbl@elt##1##2##3##4{}}%
4284      \bbl@languages
4285    \endgroup
4286  \fi
4287  \closein1
```

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

```
4288  \if/\the\toks@/\else
4289    \errhelp{language.dat loads no language, only synonyms}
4290    \errmessage{Orphan language synonym}
4291  \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.

```
4292  \let\bbl@line\@undefined
4293  \let\process@line\@undefined
4294  \let\process@synonym\@undefined
4295  \let\process@language\@undefined
4296  \let\bbl@get@enc\@undefined
4297  \let\bbl@hyph@enc\@undefined
4298  \let\bbl@tempa\@undefined
4299  \let\bbl@hook@loadkernel\@undefined
4300  \let\bbl@hook@everylanguage\@undefined
4301  \let\bbl@hook@loadpatterns\@undefined
4302  \let\bbl@hook@loadexceptions\@undefined
4303  ⟨/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].

```
4304 ⟨⟨∗More package options⟩⟩ ≡
4305 \chardef\bbl@bidimode\z@
4306 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4307 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
4308 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4309 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4310 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4311 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4312 ⟨⟨/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.

```
4313 ⟨⟨∗Font selection⟩⟩ ≡
4314 \bbl@trace{Font handling with fontspec}
4315 \ifx\ExplSyntaxOn\@undefined\else
4316   \ExplSyntaxOn
4317   \catcode`\ =10
4318   \def\bbl@loadfontspec{%
4319     \usepackage{fontspec}%  TODO. Apply patch always
4320     \expandafter
4321     \def\csname msg~text~>~fontspec/language-not-exist\endcsname##1##2##3##4{%
4322       Font '\l_fontspec_fontname_tl' is using the\\%
4323       default features for language '##1'.\\%
4324       That's usually fine, because many languages\\%
4325       require no specific features, but if the output is\\%
4326       not as expected, consider selecting another font.}
4327     \expandafter
4328     \def\csname msg~text~>~fontspec/no-script\endcsname##1##2##3##4{%
4329       Font '\l_fontspec_fontname_tl' is using the\\%
4330       default features for script '##2'.\\%
4331       That's not always wrong, but if the output is\\%
4332       not as expected, consider selecting another font.}}
4333   \ExplSyntaxOff
4334 \fi
4335 \@onlypreamble\babelfont
4336 \newcommand\babelfont[2][]{%  1=langs/scripts 2=fam
4337   \bbl@foreach{#1}{%
4338     \expandafter\ifx\csname date##1\endcsname\relax
4339       \IfFileExists{babel-##1.tex}%
4340         {\babelprovide{##1}}%
4341         {}%
4342     \fi}%
4343   \edef\bbl@tempa{#1}%
4344   \def\bbl@tempb{#2}%  Used by \bbl@bblfont
4345   \ifx\fontspec\@undefined
4346     \bbl@loadfontspec
4347   \fi
4348   \EnableBabelHook{babel-fontspec}% Just calls \bbl@switchfont
4349   \bbl@bblfont}
4350 \newcommand\bbl@bblfont[2][]{% 1=features 2=fontname, @font=rm|sf|tt
4351   \bbl@ifunset{\bbl@tempb family}%
```

```
4352      {\bbl@providefam{\bbl@tempb}}%
4353      {}%
4354  % For the default font, just in case:
4355  \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
4356  \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
4357      {\bbl@csarg\edef{\bbl@tempb dflt@}{<>{#1}{#2}}% save bbl@rmdflt@
4358        \bbl@exp{%
4359          \let\<bbl@\bbl@tempb dflt@\languagename>\<bbl@\bbl@tempb dflt@>%
4360          \\\bbl@font@set\<bbl@\bbl@tempb dflt@\languagename>%
4361                          \<\bbl@tempb default>\<\bbl@tempb family>}}%
4362      {\bbl@foreach\bbl@tempa{% ie bbl@rmdflt@lang / *scrt
4363        \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:

```
4364  \def\bbl@providefam#1{%
4365    \bbl@exp{%
4366      \\\newcommand\<#1default>{}% Just define it
4367      \\\bbl@add@list\\\bbl@font@fams{#1}%
4368      \\\DeclareRobustCommand\<#1family>{%
4369        \\\not@math@alphabet\<#1family>\relax
4370        % \\\prepare@family@series@update{#1}\<#1default>% TODO. Fails
4371        \\\fontfamily\<#1default>%
4372        \<\ifx>\\\UseHooks\\\@undefined\<\else>\\\UseHook{#1family}\<\fi>%
4373        \\\selectfont}%
4374      \\\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.

```
4375  \def\bbl@nostdfont#1{%
4376    \bbl@ifunset{bbl@WFF@\f@family}%
4377      {\bbl@csarg\gdef{WFF@\f@family}{}%  Flag, to avoid dupl warns
4378        \bbl@infowarn{The current font is not a babel standard family:\\%
4379          #1%
4380          \fontname\font\\%
4381          There is nothing intrinsically wrong with this warning, and\\%
4382          you can ignore it altogether if you do not need these\\%
4383          families. But if they are used in the document, you should be\\%
4384          aware 'babel' will no set Script and Language for them, so\\%
4385          you may consider defining a new family with \string\babelfont.\\%
4386          See the manual for further details about \string\babelfont.\\%
4387          Reported}}
4388      {}}%
4389  \gdef\bbl@switchfont{%
4390    \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
4391    \bbl@exp{%  eg Arabic -> arabic
4392      \lowercase{\edef\\\bbl@tempa{\bbl@cl{sname}}}}%
4393    \bbl@foreach\bbl@font@fams{%
4394      \bbl@ifunset{bbl@##1dflt@\languagename}%     (1) language?
4395        {\bbl@ifunset{bbl@##1dflt@*\bbl@tempa}%     (2) from script?
4396          {\bbl@ifunset{bbl@##1dflt@}%              2=F - (3) from generic?
4397            {}%                                     123=F - nothing!
4398            {\bbl@exp{%                             3=T - from generic
4399              \global\let\<bbl@##1dflt@\languagename>%
4400                        \<bbl@##1dflt@>}}%
4401          {\bbl@exp{%                               2=T - from script
4402            \global\let\<bbl@##1dflt@\languagename>%
4403                      \<bbl@##1dflt@*\bbl@tempa>}}}%
4404        {}}%                                        1=T - language, already defined
4405    \def\bbl@tempa{\bbl@nostdfont{}}%
```

```
4406   \bbl@foreach\bbl@font@fams{%       don't gather with prev for
4407     \bbl@ifunset{bbl@##1dflt@\languagename}%
4408        {\bbl@cs{famrst@##1}%
4409         \global\bbl@csarg\let{famrst@##1}\relax}%
4410        {\bbl@exp{% order is relevant. TODO: but sometimes wrong!
4411           \\\bbl@add\\\originalTeX{%
4412             \\\bbl@font@rst{\bbl@cl{##1dflt}}%
4413                           \<##1default>\<##1family>{##1}}%
4414         \\\bbl@font@set\<\bbl@##1dflt@\languagename>% the main part!
4415                          \<##1default>\<##1family>}}}%
4416   \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.

```
4417 \ifx\f@family\@undefined\else   % if latex
4418   \ifcase\bbl@engine             % if pdftex
4419     \let\bbl@ckeckstdfonts\relax
4420   \else
4421     \def\bbl@ckeckstdfonts{%
4422       \begingroup
4423         \global\let\bbl@ckeckstdfonts\relax
4424         \let\bbl@tempa\@empty
4425         \bbl@foreach\bbl@font@fams{%
4426           \bbl@ifunset{bbl@##1dflt@}%
4427             {\@nameuse{##1family}%
4428              \bbl@csarg\gdef{WFF@\f@family}{}% Flag
4429              \bbl@exp{\\\bbl@add\\\bbl@tempa{* \<##1family>= \f@family\\\\%
4430                 \space\space\fontname\font\\\\}}%
4431              \bbl@csarg\xdef{##1dflt@}{\f@family}%
4432              \expandafter\xdef\csname ##1default\endcsname{\f@family}}%
4433             {}}%
4434         \ifx\bbl@tempa\@empty\else
4435           \bbl@infowarn{The following font families will use the default\\%
4436             settings for all or some languages:\\%
4437             \bbl@tempa
4438             There is nothing intrinsically wrong with it, but\\%
4439             'babel' will no set Script and Language, which could\\%
4440              be relevant in some languages. If your document uses\\%
4441              these families, consider redefining them with \string\babelfont.\\%
4442             Reported}%
4443         \fi
4444       \endgroup}
4445   \fi
4446 \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.

```
4447 \def\bbl@font@set#1#2#3{% eg \bbl@rmdflt@lang \rmdefault \rmfamily
4448   \bbl@xin@{<>}{#1}%
4449   \ifin@
4450     \bbl@exp{\\\bbl@fontspec@set\\#1\expandafter\@gobbletwo#1\\#3}%
4451   \fi
4452   \bbl@exp{%              'Unprotected' macros return prev values
4453     \def\\#2{#1}%         eg, \rmdefault{\bbl@rmdflt@lang}
4454     \\\bbl@ifsamestring{#2}{\f@family}%
4455       {\\#3%
4456        \\\bbl@ifsamestring{\f@series}{\bfdefault}{\\\bfseries}{}%
```

162

```
4457        \let\\\bbl@tempa\relax}%
4458      {}}}
4459 %      TODO - next should be global?, but even local does its job. I'm
4460 %      still not sure -- must investigate:
4461 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
4462    \let\bbl@tempe\bbl@mapselect
4463    \let\bbl@mapselect\relax
4464    \let\bbl@temp@fam#4%        eg, '\rmfamily', to be restored below
4465    \let#4\@empty        %       Make sure \renewfontfamily is valid
4466    \bbl@exp{%
4467      \let\\\bbl@temp@pfam\<\bbl@stripslash#4\space>% eg, '\rmfamily '
4468      \<keys_if_exist:nnF>{fontspec-opentype}{Script/\bbl@cl{sname}}%
4469        {\\\newfontscript{\bbl@cl{sname}}{\bbl@cl{sotf}}}%
4470      \<keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@cl{lname}}%
4471        {\\\newfontlanguage{\bbl@cl{lname}}{\bbl@cl{lotf}}}%
4472      \\\renewfontfamily\\#4%
4473        [\bbl@cl{lsys},#2]}{#3}% ie \bbl@exp{..}{#3}
4474    \begingroup
4475      #4%
4476      \xdef#1{\f@family}%      eg, \bbl@rmdflt@lang{FreeSerif(0)}
4477    \endgroup
4478    \let#4\bbl@temp@fam
4479    \bbl@exp{\let\<\bbl@stripslash#4\space>}\bbl@temp@pfam
4480    \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.

```
4481 \def\bbl@font@rst#1#2#3#4{%
4482    \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.

```
4483 \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 :-).

```
4484 \newcommand\babelFSstore[2][]{%
4485    \bbl@ifblank{#1}%
4486      {\bbl@csarg\def{sname@#2}{Latin}}%
4487      {\bbl@csarg\def{sname@#2}{#1}}%
4488    \bbl@provide@dirs{#2}%
4489    \bbl@csarg\ifnum{wdir@#2}>\z@
4490      \let\bbl@beforeforeign\leavevmode
4491      \EnableBabelHook{babel-bidi}%
4492    \fi
4493    \bbl@foreach{#2}{%
4494      \bbl@FSstore{##1}{rm}\rmdefault\bbl@save@rmdefault
4495      \bbl@FSstore{##1}{sf}\sfdefault\bbl@save@sfdefault
4496      \bbl@FSstore{##1}{tt}\ttdefault\bbl@save@ttdefault}}
4497 \def\bbl@FSstore#1#2#3#4{%
4498    \bbl@csarg\edef{#2default#1}{#3}%
4499    \expandafter\addto\csname extras#1\endcsname{%
4500      \let#4#3%
4501      \ifx#3\f@family
4502        \edef#3{\csname bbl@#2default#1\endcsname}%
4503        \fontfamily{#3}\selectfont
4504      \else
4505        \edef#3{\csname bbl@#2default#1\endcsname}%
4506      \fi}%
```

```
4507  \expandafter\addto\csname noextras#1\endcsname{%
4508    \ifx#3\f@family
4509      \fontfamily{#4}\selectfont
4510    \fi
4511    \let#3#4}}
4512 \let\bbl@langfeatures\@empty
4513 \def\babelFSfeatures{% make sure \fontspec is redefined once
4514   \let\bbl@ori@fontspec\fontspec
4515   \renewcommand\fontspec[1][]{%
4516     \bbl@ori@fontspec[\bbl@langfeatures##1]}
4517   \let\babelFSfeatures\bbl@FSfeatures
4518   \babelFSfeatures}
4519 \def\bbl@FSfeatures#1#2{%
4520   \expandafter\addto\csname extras#1\endcsname{%
4521     \babel@save\bbl@langfeatures
4522     \edef\bbl@langfeatures{#2,}}}
4523 ⟨⟨/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.

```
4524 ⟨⟨∗Footnote changes⟩⟩ ≡
4525 \bbl@trace{Bidi footnotes}
4526 \ifnum\bbl@bidimode>\z@
4527   \def\bbl@footnote#1#2#3{%
4528     \@ifnextchar[%
4529       {\bbl@footnote@o{#1}{#2}{#3}}%
4530       {\bbl@footnote@x{#1}{#2}{#3}}}
4531   \long\def\bbl@footnote@x#1#2#3#4{%
4532     \bgroup
4533       \select@language@x{\bbl@main@language}%
4534       \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
4535     \egroup}
4536   \long\def\bbl@footnote@o#1#2#3[#4]#5{%
4537     \bgroup
4538       \select@language@x{\bbl@main@language}%
4539       \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
4540     \egroup}
4541   \def\bbl@footnotetext#1#2#3{%
4542     \@ifnextchar[%
4543       {\bbl@footnotetext@o{#1}{#2}{#3}}%
4544       {\bbl@footnotetext@x{#1}{#2}{#3}}}
4545   \long\def\bbl@footnotetext@x#1#2#3#4{%
4546     \bgroup
4547       \select@language@x{\bbl@main@language}%
4548       \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
4549     \egroup}
4550   \long\def\bbl@footnotetext@o#1#2#3[#4]#5{%
4551     \bgroup
4552       \select@language@x{\bbl@main@language}%
4553       \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
4554     \egroup}
4555   \def\BabelFootnote#1#2#3#4{%
4556     \ifx\bbl@fn@footnote\@undefined
4557       \let\bbl@fn@footnote\footnote
```

```
4558      \fi
4559      \ifx\bbl@fn@footnotetext\@undefined
4560        \let\bbl@fn@footnotetext\footnotetext
4561      \fi
4562      \bbl@ifblank{#2}%
4563        {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
4564         \@namedef{\bbl@stripslash#1text}%
4565           {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
4566        {\def#1{\bbl@exp{\\\bbl@footnote{\\\foreignlanguage{#2}}}{#3}{#4}}%
4567         \@namedef{\bbl@stripslash#1text}%
4568           {\bbl@exp{\\\bbl@footnotetext{\\\foreignlanguage{#2}}}{#3}{#4}}}}
4569 \fi
4570 ⟨⟨/Footnote changes⟩⟩
```

Now, the code.

```
4571 ⟨∗xetex⟩
4572 \def\BabelStringsDefault{unicode}
4573 \let\xebbl@stop\relax
4574 \AddBabelHook{xetex}{encodedcommands}{%
4575   \def\bbl@tempa{#1}%
4576   \ifx\bbl@tempa\@empty
4577     \XeTeXinputencoding"bytes"%
4578   \else
4579     \XeTeXinputencoding"#1"%
4580   \fi
4581   \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4582 \AddBabelHook{xetex}{stopcommands}{%
4583   \xebbl@stop
4584   \let\xebbl@stop\relax}
4585 \def\bbl@intraspace#1 #2 #3\@@{%
4586   \bbl@csarg\gdef{xeisp@\languagename}%
4587     {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4588 \def\bbl@intrapenalty#1\@@{%
4589   \bbl@csarg\gdef{xeipn@\languagename}%
4590     {\XeTeXlinebreakpenalty #1\relax}}
4591 \def\bbl@provide@intraspace{%
4592   \bbl@xin@{/s}{/\bbl@cl{lnbrk}}%
4593   \ifin@\else\bbl@xin@{/c}{/\bbl@cl{lnbrk}}\fi
4594   \ifin@
4595     \bbl@ifunset{bbl@intsp@\languagename}{}%
4596       {\expandafter\ifx\csname bbl@intsp@\languagename\endcsname\@empty\else
4597         \ifx\bbl@KVP@intraspace\@nil
4598           \bbl@exp{%
4599             \\\bbl@intraspace\bbl@cl{intsp}\\\@@}%
4600         \fi
4601         \ifx\bbl@KVP@intrapenalty\@nil
4602           \bbl@intrapenalty0\@@
4603         \fi
4604       \fi
4605     \ifx\bbl@KVP@intraspace\@nil\else % We may override the ini
4606       \expandafter\bbl@intraspace\bbl@KVP@intraspace\@@
4607     \fi
4608     \ifx\bbl@KVP@intrapenalty\@nil\else
4609       \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
4610     \fi
4611     \bbl@exp{%
4612       % TODO. Execute only once (but redundant):
4613       \\\bbl@add\<extras\languagename>{%
4614         \XeTeXlinebreaklocale "\bbl@cl{tbcp}"%
```

165

```
4615        \<bbl@xeisp@\languagename>%
4616        \<bbl@xeipn@\languagename>}%
4617      \\\bbl@toglobal\<extras\languagename>%
4618      \\\bbl@add\<noextras\languagename>{%
4619        \XeTeXlinebreaklocale "en"}%
4620      \\\bbl@toglobal\<noextras\languagename>}%
4621    \ifx\bbl@ispacesize\@undefined
4622      \gdef\bbl@ispacesize{\bbl@cl{xeisp}}%
4623      \ifx\AtBeginDocument\@notprerr
4624        \expandafter\@secondoftwo  % to execute right now
4625      \fi
4626      \AtBeginDocument{\bbl@patchfont{\bbl@ispacesize}}%
4627    \fi}%
4628  \fi}
4629 \ifx\DisableBabelHook\@undefined\endinput\fi
4630 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4631 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
4632 \DisableBabelHook{babel-fontspec}
4633 ⟨⟨Font selection⟩⟩
4634 \input txtbabel.def
4635 ⟨/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.

```
4636 ⟨∗texxet⟩
4637 \providecommand\bbl@provide@intraspace{}
4638 \bbl@trace{Redefinitions for bidi layout}
4639 \def\bbl@sspre@caption{%
4640  \bbl@exp{\everyhbox{\\\bbl@textdir\bbl@cs{wdir@\bbl@main@language}}}}
4641 \ifx\bbl@opt@layout\@nnil\endinput\fi  % No layout
4642 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
4643 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
4644 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
4645  \def\@hangfrom#1{%
4646    \setbox\@tempboxa\hbox{{#1}}%
4647    \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
4648    \noindent\box\@tempboxa}
4649  \def\raggedright{%
4650    \let\\\@centercr
4651    \bbl@startskip\z@skip
4652    \@rightskip\@flushglue
4653    \bbl@endskip\@rightskip
4654    \parindent\z@
4655    \parfillskip\bbl@startskip}
4656  \def\raggedleft{%
4657    \let\\\@centercr
4658    \bbl@startskip\@flushglue
4659    \bbl@endskip\z@skip
4660    \parindent\z@
4661    \parfillskip\bbl@endskip}
4662 \fi
```

```
4663 \IfBabelLayout{lists}
4664   {\bbl@sreplace\list
4665     {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
4666   \def\bbl@listleftmargin{%
4667     \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
4668   \ifcase\bbl@engine
4669     \def\labelenumii{)\theenumii(}% pdftex doesn't reverse ()
4670     \def\p@enumiii{\p@enumii)\theenumii(}%
4671   \fi
4672   \bbl@sreplace\@verbatim
4673     {\leftskip\@totalleftmargin}%
4674     {\bbl@startskip\textwidth
4675      \advance\bbl@startskip-\linewidth}%
4676   \bbl@sreplace\@verbatim
4677     {\rightskip\z@skip}%
4678     {\bbl@endskip\z@skip}}%
4679   {}
4680 \IfBabelLayout{contents}
4681   {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
4682    \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
4683   {}
4684 \IfBabelLayout{columns}
4685   {\bbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbl@outputhbox}%
4686   \def\bbl@outputhbox#1{%
4687     \hb@xt@\textwidth{%
4688       \hskip\columnwidth
4689       \hfil
4690       {\normalcolor\vrule \@width\columnseprule}%
4691       \hfil
4692       \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
4693       \hskip-\textwidth
4694       \hb@xt@\columnwidth{\box\@outputbox \hss}%
4695       \hskip\columnsep
4696       \hskip\columnwidth}}}%
4697   {}
4698 ⟨⟨Footnote changes⟩⟩
4699 \IfBabelLayout{footnotes}%
4700   {\BabelFootnote\footnote\languagename{}{}%
4701    \BabelFootnote\localfootnote\languagename{}{}%
4702    \BabelFootnote\mainfootnote{}{}{}}
4703   {}
```

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.

```
4704 \IfBabelLayout{counters}%
4705   {\let\bbl@latinarabic=\@arabic
4706    \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
4707    \let\bbl@asciiroman=\@roman
4708    \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciiroman#1}}}%
4709    \let\bbl@asciiRoman=\@Roman
4710    \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}}{}
4711 ⟨/texxet⟩
```

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).

```
4712 ⟨*luatex⟩
4713 \ifx\AddBabelHook\@undefined % When plain.def, babel.sty starts
4714 \bbl@trace{Read language.dat}
4715 \ifx\bbl@readstream\@undefined
4716   \csname newread\endcsname\bbl@readstream
4717 \fi
4718 \begingroup
4719   \toks@{}
4720   \count@\z@ % 0=start, 1=0th, 2=normal
4721   \def\bbl@process@line#1#2 #3 #4 {%
4722     \ifx=#1%
4723       \bbl@process@synonym{#2}%
4724     \else
4725       \bbl@process@language{#1#2}{#3}{#4}%
4726     \fi
4727     \ignorespaces}
4728   \def\bbl@manylang{%
4729     \ifnum\bbl@last>\@ne
4730       \bbl@info{Non-standard hyphenation setup}%
4731     \fi
4732     \let\bbl@manylang\relax}
4733   \def\bbl@process@language#1#2#3{%
4734     \ifcase\count@
4735       \@ifundefined{zth@#1}{\count@\tw@}{\count@\@ne}%
4736     \or
4737       \count@\tw@
4738     \fi
4739     \ifnum\count@=\tw@
4740       \expandafter\addlanguage\csname l@#1\endcsname
4741       \language\allocationnumber
```

168

```
4742      \chardef\bbl@last\allocationnumber
4743      \bbl@manylang
4744      \let\bbl@elt\relax
4745      \xdef\bbl@languages{%
4746         \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
4747    \fi
4748    \the\toks@
4749    \toks@{}}
4750 \def\bbl@process@synonym@aux#1#2{%
4751    \global\expandafter\chardef\csname l@#1\endcsname#2\relax
4752    \let\bbl@elt\relax
4753    \xdef\bbl@languages{%
4754       \bbl@languages\bbl@elt{#1}{#2}{}{}}}%
4755 \def\bbl@process@synonym#1{%
4756    \ifcase\count@
4757       \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
4758    \or
4759       \@ifundefined{zth@#1}{\bbl@process@synonym@aux{#1}{0}}{}%
4760    \else
4761       \bbl@process@synonym@aux{#1}{\the\bbl@last}%
4762    \fi}
4763 \ifx\bbl@languages\@undefined % Just a (sensible?) guess
4764    \chardef\l@english\z@
4765    \chardef\l@USenglish\z@
4766    \chardef\bbl@last\z@
4767    \global\@namedef{bbl@hyphendata@0}{{hyphen.tex}{}}
4768    \gdef\bbl@languages{%
4769       \bbl@elt{english}{0}{hyphen.tex}{}%
4770       \bbl@elt{USenglish}{0}{}{}}
4771 \else
4772    \global\let\bbl@languages@format\bbl@languages
4773    \def\bbl@elt#1#2#3#4{% Remove all except language 0
4774       \ifnum#2>\z@\else
4775          \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
4776       \fi}%
4777    \xdef\bbl@languages{\bbl@languages}%
4778 \fi
4779 \def\bbl@elt#1#2#3#4{\@namedef{zth@#1}{}} % Define flags
4780 \bbl@languages
4781 \openin\bbl@readstream=language.dat
4782 \ifeof\bbl@readstream
4783    \bbl@warning{I couldn't find language.dat. No additional\\%
4784                 patterns loaded. Reported}%
4785 \else
4786    \loop
4787       \endlinechar\m@ne
4788       \read\bbl@readstream to \bbl@line
4789       \endlinechar`\^^M
4790       \if T\ifeof\bbl@readstream F\fi T\relax
4791          \ifx\bbl@line\@empty\else
4792             \edef\bbl@line{\bbl@line\space\space\space}%
4793             \expandafter\bbl@process@line\bbl@line\relax
4794          \fi
4795    \repeat
4796 \fi
4797 \endgroup
4798 \bbl@trace{Macros for reading patterns files}
4799 \def\bbl@get@enc#1:#2:#3\@@@{\def\bbl@hyph@enc{#2}}
4800 \ifx\babelcatcodetablenum\@undefined
```

```
4801    \ifx\newcatcodetable\@undefined
4802      \def\babelcatcodetablenum{5211}
4803      \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4804    \else
4805      \newcatcodetable\babelcatcodetablenum
4806      \newcatcodetable\bbl@pattcodes
4807    \fi
4808 \else
4809   \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4810 \fi
4811 \def\bbl@luapatterns#1#2{%
4812   \bbl@get@enc#1::\@@@
4813   \setbox\z@\hbox\bgroup
4814     \begingroup
4815       \savecatcodetable\babelcatcodetablenum\relax
4816       \initcatcodetable\bbl@pattcodes\relax
4817       \catcodetable\bbl@pattcodes\relax
4818         \catcode`\#=6  \catcode`\$=3 \catcode`\&=4 \catcode`\^=7
4819         \catcode`\_=8  \catcode`\{=1 \catcode`\}=2 \catcode`\~=13
4820         \catcode`\@=11 \catcode`\^^I=10 \catcode`\^^J=12
4821         \catcode`\<=12 \catcode`\>=12 \catcode`\*=12 \catcode`\.=12
4822         \catcode`\-=12 \catcode`\/=12 \catcode`\[=12 \catcode`\]=12
4823         \catcode`\`=12 \catcode`\'=12 \catcode`\"=12
4824         \input #1\relax
4825       \catcodetable\babelcatcodetablenum\relax
4826     \endgroup
4827     \def\bbl@tempa{#2}%
4828     \ifx\bbl@tempa\@empty\else
4829       \input #2\relax
4830     \fi
4831   \egroup}%
4832 \def\bbl@patterns@lua#1{%
4833   \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
4834     \csname l@#1\endcsname
4835     \edef\bbl@tempa{#1}%
4836   \else
4837     \csname l@#1:\f@encoding\endcsname
4838     \edef\bbl@tempa{#1:\f@encoding}%
4839   \fi\relax
4840   \@namedef{lu@texhyphen@loaded@\the\language}{}% Temp
4841   \@ifundefined{bbl@hyphendata@\the\language}%
4842     {\def\bbl@elt##1##2##3##4{%
4843       \ifnum##2=\csname l@\bbl@tempa\endcsname % #2=spanish, dutch:OT1...
4844         \def\bbl@tempb{##3}%
4845         \ifx\bbl@tempb\@empty\else % if not a synonymous
4846           \def\bbl@tempc{{##3}{##4}}%
4847         \fi
4848         \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
4849       \fi}%
4850     \bbl@languages
4851     \@ifundefined{bbl@hyphendata@\the\language}%
4852       {\bbl@info{No hyphenation patterns were set for\\%
4853                  language '\bbl@tempa'. Reported}}%
4854       {\expandafter\expandafter\expandafter\bbl@luapatterns
4855         \csname bbl@hyphendata@\the\language\endcsname}}{}}
4856 \endinput\fi
4857   % Here ends \ifx\AddBabelHook\@undefined
4858   % A few lines are only read by hyphen.cfg
4859 \ifx\DisableBabelHook\@undefined
```

```
4860  \AddBabelHook{luatex}{everylanguage}{%
4861    \def\process@language##1##2##3{%
4862      \def\process@line####1####2 ####3 ####4 {}}}
4863  \AddBabelHook{luatex}{loadpatterns}{%
4864    \input #1\relax
4865    \expandafter\gdef\csname bbl@hyphendata@\the\language\endcsname
4866      {{#1}{}}}
4867  \AddBabelHook{luatex}{loadexceptions}{%
4868    \input #1\relax
4869    \def\bbl@tempb##1##2{{##1}{#1}}%
4870    \expandafter\xdef\csname bbl@hyphendata@\the\language\endcsname
4871      {\expandafter\expandafter\expandafter\bbl@tempb
4872        \csname bbl@hyphendata@\the\language\endcsname}}
4873 \endinput\fi
4874  % Here stops reading code for hyphen.cfg
4875  % The following is read the 2nd time it's loaded
4876 \begingroup  % TODO - to a lua file
4877 \catcode`\%=12
4878 \catcode`\'=12
4879 \catcode`\"=12
4880 \catcode`\:=12
4881 \directlua{
4882   Babel = Babel or {}
4883   function Babel.bytes(line)
4884     return line:gsub("(.)",
4885       function (chr) return unicode.utf8.char(string.byte(chr)) end)
4886   end
4887   function Babel.begin_process_input()
4888     if luatexbase and luatexbase.add_to_callback then
4889       luatexbase.add_to_callback('process_input_buffer',
4890                                   Babel.bytes,'Babel.bytes')
4891     else
4892       Babel.callback = callback.find('process_input_buffer')
4893       callback.register('process_input_buffer',Babel.bytes)
4894     end
4895   end
4896   function Babel.end_process_input ()
4897     if luatexbase and luatexbase.remove_from_callback then
4898       luatexbase.remove_from_callback('process_input_buffer','Babel.bytes')
4899     else
4900       callback.register('process_input_buffer',Babel.callback)
4901     end
4902   end
4903   function Babel.addpatterns(pp, lg)
4904     local lg = lang.new(lg)
4905     local pats = lang.patterns(lg) or ''
4906     lang.clear_patterns(lg)
4907     for p in pp:gmatch('[^%s]+') do
4908       ss = ''
4909       for i in string.utfcharacters(p:gsub('%d', '')) do
4910         ss = ss .. '%d?' .. i
4911       end
4912       ss = ss:gsub('^%%d%?%.', '%%.') .. '%d?'
4913       ss = ss:gsub('%.%%d%?$', '%%.')
4914       pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
4915       if n == 0 then
4916         tex.sprint(
4917           [[\string\csname\space bbl@info\endcsname{New pattern: ]]
4918           .. p .. [[}]])
```

171

```
4919        pats = pats .. ' ' .. p
4920      else
4921        tex.sprint(
4922          [[\string\csname\space bbl@info\endcsname{Renew pattern: ]]
4923          .. p .. [[}]])
4924      end
4925    end
4926    lang.patterns(lg, pats)
4927  end
4928 }
4929 \endgroup
4930 \ifx\newattribute\@undefined\else
4931   \newattribute\bbl@attr@locale
4932   \directlua{ Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale' }
4933   \AddBabelHook{luatex}{beforeextras}{%
4934     \setattribute\bbl@attr@locale\localeid}
4935 \fi
4936 \def\BabelStringsDefault{unicode}
4937 \let\luabbl@stop\relax
4938 \AddBabelHook{luatex}{encodedcommands}{%
4939   \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
4940   \ifx\bbl@tempa\bbl@tempb\else
4941     \directlua{Babel.begin_process_input()}%
4942     \def\luabbl@stop{%
4943       \directlua{Babel.end_process_input()}}%
4944   \fi}%
4945 \AddBabelHook{luatex}{stopcommands}{%
4946   \luabbl@stop
4947   \let\luabbl@stop\relax}
4948 \AddBabelHook{luatex}{patterns}{%
4949   \@ifundefined{bbl@hyphendata@\the\language}%
4950     {\def\bbl@elt##1##2##3##4{%
4951       \ifnum##2=\csname l@##1\endcsname % #2=spanish, dutch:OT1...
4952         \def\bbl@tempb{##3}%
4953         \ifx\bbl@tempb\@empty\else % if not a synonymous
4954           \def\bbl@tempc{{##3}{##4}}%
4955         \fi
4956         \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
4957       \fi}%
4958     \bbl@languages
4959     \@ifundefined{bbl@hyphendata@\the\language}%
4960       {\bbl@info{No hyphenation patterns were set for\\%
4961                 language '#2'. Reported}}%
4962       {\expandafter\expandafter\expandafter\bbl@luapatterns
4963         \csname bbl@hyphendata@\the\language\endcsname}}{}%
4964   \@ifundefined{bbl@patterns@}{}{%
4965     \begingroup
4966       \bbl@xin@{,\number\language,}{,\bbl@pttnlist}%
4967       \ifin@\else
4968         \ifx\bbl@patterns@\@empty\else
4969           \directlua{ Babel.addpatterns(
4970             [[\bbl@patterns@]], \number\language) }%
4971         \fi
4972         \@ifundefined{bbl@patterns@#1}%
4973           \@empty
4974           {\directlua{ Babel.addpatterns(
4975             [[\space\csname bbl@patterns@#1\endcsname]],
4976             \number\language) }}%
4977         \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
```

```
4978        \fi
4979      \endgroup}%
4980   \bbl@exp{%
4981     \bbl@ifunset{bbl@prehc@\languagename}{}%
4982       {\\\bbl@ifblank{\bbl@cs{prehc@\languagename}}{}%
4983         {\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.

```
4984 \@onlypreamble\babelpatterns
4985 \AtEndOfPackage{%
4986   \newcommand\babelpatterns[2][\@empty]{%
4987     \ifx\bbl@patterns@\relax
4988       \let\bbl@patterns@\@empty
4989     \fi
4990     \ifx\bbl@pttnlist\@empty\else
4991       \bbl@warning{%
4992         You must not intermingle \string\selectlanguage\space and\\%
4993         \string\babelpatterns\space or some patterns will not\\%
4994         be taken into account. Reported}%
4995     \fi
4996     \ifx\@empty#1%
4997       \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
4998     \else
4999       \edef\bbl@tempb{\zap@space#1 \@empty}%
5000       \bbl@for\bbl@tempa\bbl@tempb{%
5001         \bbl@fixname\bbl@tempa
5002         \bbl@iflanguage\bbl@tempa{%
5003           \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
5004             \@ifundefined{bbl@patterns@\bbl@tempa}%
5005               \@empty
5006               {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
5007             #2}}}%
5008     \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.

```
5009 % TODO - to a lua file
5010 \directlua{
5011   Babel = Babel or {}
5012   Babel.linebreaking = Babel.linebreaking or {}
5013   Babel.linebreaking.before = {}
5014   Babel.linebreaking.after = {}
5015   Babel.locale = {} % Free to use, indexed by \localeid
5016   function Babel.linebreaking.add_before(func)
5017     tex.print([[\noexpand\csname bbl@luahyphenate\endcsname]])
5018     table.insert(Babel.linebreaking.before, func)
5019   end
5020   function Babel.linebreaking.add_after(func)
5021     tex.print([[\noexpand\csname bbl@luahyphenate\endcsname]])
5022     table.insert(Babel.linebreaking.after, func)
5023   end
5024 }
```

```
5025 \def\bbl@intraspace#1 #2 #3\@@{%
5026   \directlua{
5027     Babel = Babel or {}
5028     Babel.intraspaces = Babel.intraspaces or {}
5029     Babel.intraspaces['\csname bbl@sbcp@\languagename\endcsname'] = %
5030       {b = #1, p = #2, m = #3}
5031     Babel.locale_props[\the\localeid].intraspace = %
5032       {b = #1, p = #2, m = #3}
5033   }}
5034 \def\bbl@intrapenalty#1\@@{%
5035   \directlua{
5036     Babel = Babel or {}
5037     Babel.intrapenalties = Babel.intrapenalties or {}
5038     Babel.intrapenalties['\csname bbl@sbcp@\languagename\endcsname'] = #1
5039     Babel.locale_props[\the\localeid].intrapenalty = #1
5040   }}
5041 \begingroup
5042 \catcode`\%=12
5043 \catcode`\^=14
5044 \catcode`\'=12
5045 \catcode`\~=12
5046 \gdef\bbl@seaintraspace{^
5047   \let\bbl@seaintraspace\relax
5048   \directlua{
5049     Babel = Babel or {}
5050     Babel.sea_enabled = true
5051     Babel.sea_ranges = Babel.sea_ranges or {}
5052     function Babel.set_chranges (script, chrng)
5053       local c = 0
5054       for s, e in string.gmatch(chrng..' ', '(.-)%.%.(.-)%s') do
5055         Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
5056         c = c + 1
5057       end
5058     end
5059     function Babel.sea_disc_to_space (head)
5060       local sea_ranges = Babel.sea_ranges
5061       local last_char = nil
5062       local quad = 655360        ^% 10 pt = 655360 = 10 * 65536
5063       for item in node.traverse(head) do
5064         local i = item.id
5065         if i == node.id'glyph' then
5066           last_char = item
5067         elseif i == 7 and item.subtype == 3 and last_char
5068             and last_char.char > 0x0C99 then
5069           quad = font.getfont(last_char.font).size
5070           for lg, rg in pairs(sea_ranges) do
5071             if last_char.char > rg[1] and last_char.char < rg[2] then
5072               lg = lg:sub(1, 4)  ^% Remove trailing number of, eg, Cyrl1
5073               local intraspace = Babel.intraspaces[lg]
5074               local intrapenalty = Babel.intrapenalties[lg]
5075               local n
5076               if intrapenalty ~= 0 then
5077                 n = node.new(14, 0)      ^% penalty
5078                 n.penalty = intrapenalty
5079                 node.insert_before(head, item, n)
5080               end
5081               n = node.new(12, 13)      ^% (glue, spaceskip)
5082               node.setglue(n, intraspace.b * quad,
5083                               intraspace.p * quad,
```

174

```
5084                              intraspace.m * quad)
5085                 node.insert_before(head, item, n)
5086                 node.remove(head, item)
5087               end
5088             end
5089           end
5090         end
5091       end
5092     }^^
5093   \bbl@luahyphenate}
```

13.5   CJK line breaking

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

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

```
5094 \catcode`\%=14
5095 \gdef\bbl@cjkintraspace{%
5096   \let\bbl@cjkintraspace\relax
5097   \directlua{
5098     Babel = Babel or {}
5099     require('babel-data-cjk.lua')
5100     Babel.cjk_enabled = true
5101     function Babel.cjk_linebreak(head)
5102       local GLYPH = node.id'glyph'
5103       local last_char = nil
5104       local quad = 655360      % 10 pt = 655360 = 10 * 65536
5105       local last_class = nil
5106       local last_lang = nil
5107
5108       for item in node.traverse(head) do
5109         if item.id == GLYPH then
5110
5111           local lang = item.lang
5112
5113           local LOCALE = node.get_attribute(item,
5114                 Babel.attr_locale)
5115           local props = Babel.locale_props[LOCALE]
5116
5117           local class = Babel.cjk_class[item.char].c
5118
5119           if props.cjk_quotes and props.cjk_quotes[item.char] then
5120             class = props.cjk_quotes[item.char]
5121           end
5122
5123           if class == 'cp' then class = 'cl' end % )] as CL
5124           if class == 'id' then class = 'I' end
5125
5126           local br = 0
5127           if class and last_class and Babel.cjk_breaks[last_class][class] then
5128             br = Babel.cjk_breaks[last_class][class]
5129           end
5130
5131           if br == 1 and props.linebreak == 'c' and
5132                 lang ~= \the\l@nohyphenation\space and
```

```
5133            last_lang ~= \the\l@nohyphenation then
5134          local intrapenalty = props.intrapenalty
5135          if intrapenalty ~= 0 then
5136            local n = node.new(14, 0)       % penalty
5137            n.penalty = intrapenalty
5138            node.insert_before(head, item, n)
5139          end
5140          local intraspace = props.intraspace
5141          local n = node.new(12, 13)        % (glue, spaceskip)
5142          node.setglue(n, intraspace.b * quad,
5143                            intraspace.p * quad,
5144                            intraspace.m * quad)
5145          node.insert_before(head, item, n)
5146        end
5147
5148        if font.getfont(item.font) then
5149          quad = font.getfont(item.font).size
5150        end
5151        last_class = class
5152        last_lang = lang
5153      else % if penalty, glue or anything else
5154        last_class = nil
5155      end
5156    end
5157    lang.hyphenate(head)
5158  end
5159 }%
5160 \bbl@luahyphenate}
5161 \gdef\bbl@luahyphenate{%
5162  \let\bbl@luahyphenate\relax
5163  \directlua{
5164    luatexbase.add_to_callback('hyphenate',
5165    function (head, tail)
5166      if Babel.linebreaking.before then
5167        for k, func in ipairs(Babel.linebreaking.before)  do
5168          func(head)
5169        end
5170      end
5171      if Babel.cjk_enabled then
5172        Babel.cjk_linebreak(head)
5173      end
5174      lang.hyphenate(head)
5175      if Babel.linebreaking.after then
5176        for k, func in ipairs(Babel.linebreaking.after)  do
5177          func(head)
5178        end
5179      end
5180      if Babel.sea_enabled then
5181        Babel.sea_disc_to_space(head)
5182      end
5183    end,
5184    'Babel.hyphenate')
5185  }
5186 }
5187 \endgroup
5188 \def\bbl@provide@intraspace{%
5189  \bbl@ifunset{bbl@intsp@\languagename}{}%
5190    {\expandafter\ifx\csname bbl@intsp@\languagename\endcsname\@empty\else
5191      \bbl@xin@{/c}{/\bbl@cl{lnbrk}}%
```

```
5192        \ifin@          % cjk
5193          \bbl@cjkintraspace
5194          \directlua{
5195              Babel = Babel or {}
5196              Babel.locale_props = Babel.locale_props or {}
5197              Babel.locale_props[\the\localeid].linebreak = 'c'
5198          }%
5199          \bbl@exp{\\\bbl@intraspace\bbl@cl{intsp}\\\@@}%
5200          \ifx\bbl@KVP@intrapenalty\@nil
5201            \bbl@intrapenalty0\@@
5202          \fi
5203        \else            % sea
5204          \bbl@seaintraspace
5205          \bbl@exp{\\\bbl@intraspace\bbl@cl{intsp}\\\@@}%
5206          \directlua{
5207            Babel = Babel or {}
5208            Babel.sea_ranges = Babel.sea_ranges or {}
5209            Babel.set_chranges('\bbl@cl{sbcp}',
5210                               '\bbl@cl{chrng}')
5211          }%
5212          \ifx\bbl@KVP@intrapenalty\@nil
5213            \bbl@intrapenalty0\@@
5214          \fi
5215        \fi
5216      \fi
5217      \ifx\bbl@KVP@intrapenalty\@nil\else
5218        \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
5219      \fi}}
```

13.6 Arabic justification

```
5220 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5221 \def\bblar@chars{%
5222   0628,0629,062A,062B,062C,062D,062E,062F,0630,0631,0632,0633,%
5223   0634,0635,0636,0637,0638,0639,063A,063B,063C,063D,063E,063F,%
5224   0640,0641,0642,0643,0644,0645,0646,0647,0649}
5225 \def\bblar@elongated{%
5226   0626,0628,062A,062B,0633,0634,0635,0636,063B,%
5227   063C,063D,063E,063F,0641,0642,0643,0644,0646,%
5228   0649,064A}
5229 \begingroup
5230   \catcode`\_=11 \catcode`:=11
5231   \gdef\bblar@nofswarn{\gdef\msg_warning:nnx##1##2##3{}}
5232 \endgroup
5233 \gdef\bbl@arabicjust{%
5234   \let\bbl@arabicjust\relax
5235   \newattribute\bblar@kashida
5236   \directlua{ Babel.attr_kashida = luatexbase.registernumber'bblar@kashida' }%
5237   \bblar@kashida=\z@
5238   \bbl@patchfont{{\bbl@parsejalt}}%
5239   \directlua{
5240     Babel.arabic.elong_map   = Babel.arabic.elong_map or {}
5241     Babel.arabic.elong_map[\the\localeid]   = {}
5242     luatexbase.add_to_callback('post_linebreak_filter',
5243       Babel.arabic.justify, 'Babel.arabic.justify')
5244     luatexbase.add_to_callback('hpack_filter',
5245       Babel.arabic.justify_hbox, 'Babel.arabic.justify_hbox')
5246   }}%
5247 % Save both node lists to make replacement. TODO. Save also widths to
```

```
5248 % make computations
5249 \def\bblar@fetchjalt#1#2#3#4{%
5250   \bbl@exp{\\\bbl@foreach{#1}}{%
5251     \bbl@ifunset{bblar@JE@##1}%
5252       {\setbox\z@\hbox{^^^^200d\char"##1#2}}%
5253       {\setbox\z@\hbox{^^^^200d\char"\@nameuse{bblar@JE@##1}#2}}%
5254     \directlua{%
5255       local last = nil
5256       for item in node.traverse(tex.box[0].head) do
5257         if item.id == node.id'glyph' and item.char > 0x600 and
5258             not (item.char == 0x200D) then
5259           last = item
5260         end
5261       end
5262       Babel.arabic.#3['##1#4'] = last.char
5263     }}}
5264 % Brute force. No rules at all, yet. The ideal: look at jalt table. And
5265 % perhaps other tables (falt?, cswh?). What about kaf? And diacritic
5266 % positioning?
5267 \gdef\bbl@parsejalt{%
5268   \ifx\addfontfeature\@undefined\else
5269   \bbl@xin@{/e}{/\bbl@cl{lnbrk}}%
5270   \ifin@
5271     \directlua{%
5272       if Babel.arabic.elong_map[\the\localeid][\fontid\font] == nil then
5273         Babel.arabic.elong_map[\the\localeid][\fontid\font] = {}
5274         tex.print([[\string\csname\space bbl@parsejalti\endcsname]])
5275       end
5276     }%
5277   \fi
5278 \fi}
5279 \gdef\bbl@parsejalti{%
5280   \begingroup
5281     \let\bbl@parsejalt\relax    % To avoid infinite loop
5282     \edef\bbl@tempb{\fontid\font}%
5283     \bblar@nofswarn
5284     \bblar@fetchjalt\bblar@elongated{}{from}{}%
5285     \bblar@fetchjalt\bblar@chars{^^^^064a}{from}{a}% Alef maksura
5286     \bblar@fetchjalt\bblar@chars{^^^^0649}{from}{y}% Yeh
5287     \addfontfeature{RawFeature=+jalt}%
5288     % \@namedef{bblar@JE@0643}{06AA}% todo: catch medial kaf
5289     \bblar@fetchjalt\bblar@elongated{}{dest}{}%
5290     \bblar@fetchjalt\bblar@chars{^^^^064a}{dest}{a}%
5291     \bblar@fetchjalt\bblar@chars{^^^^0649}{dest}{y}%
5292     \directlua{%
5293       for k, v in pairs(Babel.arabic.from) do
5294         if Babel.arabic.dest[k] and
5295             not (Babel.arabic.from[k] == Babel.arabic.dest[k]) then
5296           Babel.arabic.elong_map[\the\localeid][\bbl@tempb]
5297             [Babel.arabic.from[k]] = Babel.arabic.dest[k]
5298         end
5299       end
5300     }%
5301   \endgroup}
5302 %
5303 \begingroup
5304 \catcode`\#=11
5305 \catcode`\~=11
5306 \directlua{
```

```lua
5307
5308 Babel.arabic = Babel.arabic or {}
5309 Babel.arabic.from = {}
5310 Babel.arabic.dest = {}
5311 Babel.arabic.justify_factor = 0.95
5312 Babel.arabic.justify_enabled = true
5313
5314 function Babel.arabic.justify(head)
5315   if not Babel.arabic.justify_enabled then return head end
5316   for line in node.traverse_id(node.id'hlist', head) do
5317     Babel.arabic.justify_hlist(head, line)
5318   end
5319   return head
5320 end
5321
5322 function Babel.arabic.justify_hbox(head, gc, size, pack)
5323   local has_inf = false
5324   if Babel.arabic.justify_enabled and pack == 'exactly' then
5325     for n in node.traverse_id(12, head) do
5326       if n.stretch_order > 0 then has_inf = true end
5327     end
5328     if not has_inf then
5329       Babel.arabic.justify_hlist(head, nil, gc, size, pack)
5330     end
5331   end
5332   return head
5333 end
5334
5335 function Babel.arabic.justify_hlist(head, line, gc, size, pack)
5336   local d, new
5337   local k_list, k_item, pos_inline
5338   local width, width_new, full, k_curr, wt_pos, goal, shift
5339   local subst_done = false
5340   local elong_map = Babel.arabic.elong_map
5341   local last_line
5342   local GLYPH = node.id'glyph'
5343   local KASHIDA = Babel.attr_kashida
5344   local LOCALE = Babel.attr_locale
5345
5346   if line == nil then
5347     line = {}
5348     line.glue_sign = 1
5349     line.glue_order = 0
5350     line.head = head
5351     line.shift = 0
5352     line.width = size
5353   end
5354
5355   % Exclude last line. todo. But-- it discards one-word lines, too!
5356   % ? Look for glue = 12:15
5357   if (line.glue_sign == 1 and line.glue_order == 0) then
5358     elongs = {}      % Stores elongated candidates of each line
5359     k_list = {}      % And all letters with kashida
5360     pos_inline = 0   % Not yet used
5361
5362     for n in node.traverse_id(GLYPH, line.head) do
5363       pos_inline = pos_inline + 1 % To find where it is. Not used.
5364
5365       % Elongated glyphs
```

```
5366        if elong_map then
5367          local locale = node.get_attribute(n, LOCALE)
5368          if elong_map[locale] and elong_map[locale][n.font] and
5369              elong_map[locale][n.font][n.char] then
5370            table.insert(elongs, {node = n, locale = locale} )
5371            node.set_attribute(n.prev, KASHIDA, 0)
5372          end
5373        end
5374
5375        % Tatwil
5376        if Babel.kashida_wts then
5377          local k_wt = node.get_attribute(n, KASHIDA)
5378          if k_wt > 0 then % todo. parameter for multi inserts
5379            table.insert(k_list, {node = n, weight = k_wt, pos = pos_inline})
5380          end
5381        end
5382
5383      end % of node.traverse_id
5384
5385      if #elongs == 0 and #k_list == 0 then goto next_line end
5386      full  = line.width
5387      shift = line.shift
5388      goal  = full * Babel.arabic.justify_factor % A bit crude
5389      width = node.dimensions(line.head)     % The 'natural' width
5390
5391      % == Elongated ==
5392      % Original idea taken from 'chikenize'
5393      while (#elongs > 0 and width < goal) do
5394        subst_done = true
5395        local x = #elongs
5396        local curr = elongs[x].node
5397        local oldchar = curr.char
5398        curr.char = elong_map[elongs[x].locale][curr.font][curr.char]
5399        width = node.dimensions(line.head)  % Check if the line is too wide
5400        % Substitute back if the line would be too wide and break:
5401        if width > goal then
5402          curr.char = oldchar
5403          break
5404        end
5405        % If continue, pop the just substituted node from the list:
5406        table.remove(elongs, x)
5407      end
5408
5409      % == Tatwil ==
5410      if #k_list == 0 then goto next_line end
5411
5412      width = node.dimensions(line.head)     % The 'natural' width
5413      k_curr = #k_list
5414      wt_pos = 1
5415
5416      while width < goal do
5417        subst_done = true
5418        k_item = k_list[k_curr].node
5419        if k_list[k_curr].weight == Babel.kashida_wts[wt_pos] then
5420          d = node.copy(k_item)
5421          d.char = 0x0640
5422          line.head, new = node.insert_after(line.head, k_item, d)
5423          width_new = node.dimensions(line.head)
5424          if width > goal or width == width_new then
```

```
5425        node.remove(line.head, new) % Better compute before
5426        break
5427      end
5428    width = width_new
5429    end
5430    if k_curr == 1 then
5431      k_curr = #k_list
5432      wt_pos = (wt_pos >= table.getn(Babel.kashida_wts)) and 1 or wt_pos+1
5433    else
5434      k_curr = k_curr - 1
5435    end
5436  end
5437
5438  ::next_line::
5439
5440  % Must take into account marks and ins, see luatex manual.
5441  % Have to be executed only if there are changes. Investigate
5442  % what's going on exactly.
5443  if subst_done and not gc then
5444    d = node.hpack(line.head, full, 'exactly')
5445    d.shift = shift
5446    node.insert_before(head, line, d)
5447    node.remove(head, line)
5448  end
5449  end % if process line
5450 end
5451 }
5452 \endgroup
5453 \fi\fi % Arabic just block
```

13.7 Common stuff

```
5454 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
5455 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
5456 \DisableBabelHook{babel-fontspec}
5457 ⟨⟨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.

```
5458 % TODO - to a lua file
5459 \directlua{
5460 Babel.script_blocks = {
5461   ['dflt'] = {},
5462   ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
5463              {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
5464   ['Armn'] = {{0x0530, 0x058F}},
5465   ['Beng'] = {{0x0980, 0x09FF}},
5466   ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
5467   ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
5468   ['Cyrl'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
5469              {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
5470   ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
5471   ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
```

```
5472              {0xAB00, 0xAB2F}},
5473   ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
5474   % Don't follow strictly Unicode, which places some Coptic letters in
5475   % the 'Greek and Coptic' block
5476   ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
5477   ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
5478              {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
5479              {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
5480              {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
5481              {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
5482              {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
5483   ['Hebr'] = {{0x0590, 0x05FF}},
5484   ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
5485              {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
5486   ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
5487   ['Knda'] = {{0x0C80, 0x0CFF}},
5488   ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
5489              {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
5490              {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
5491   ['Laoo'] = {{0x0E80, 0x0EFF}},
5492   ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
5493              {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
5494              {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
5495   ['Mahj'] = {{0x11150, 0x1117F}},
5496   ['Mlym'] = {{0x0D00, 0x0D7F}},
5497   ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
5498   ['Orya'] = {{0x0B00, 0x0B7F}},
5499   ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
5500   ['Syrc'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
5501   ['Taml'] = {{0x0B80, 0x0BFF}},
5502   ['Telu'] = {{0x0C00, 0x0C7F}},
5503   ['Tfng'] = {{0x2D30, 0x2D7F}},
5504   ['Thai'] = {{0x0E00, 0x0E7F}},
5505   ['Tibt'] = {{0x0F00, 0x0FFF}},
5506   ['Vaii'] = {{0xA500, 0xA63F}},
5507   ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
5508 }
5509
5510 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
5511 Babel.script_blocks.Hant = Babel.script_blocks.Hans
5512 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
5513
5514 function Babel.locale_map(head)
5515   if not Babel.locale_mapped then return head end
5516
5517   local LOCALE = Babel.attr_locale
5518   local GLYPH = node.id('glyph')
5519   local inmath = false
5520   local toloc_save
5521   for item in node.traverse(head) do
5522     local toloc
5523     if not inmath and item.id == GLYPH then
5524       % Optimization: build a table with the chars found
5525       if Babel.chr_to_loc[item.char] then
5526         toloc = Babel.chr_to_loc[item.char]
5527       else
5528         for lc, maps in pairs(Babel.loc_to_scr) do
5529           for _, rg in pairs(maps) do
5530             if item.char >= rg[1] and item.char <= rg[2] then
```

```
5531            Babel.chr_to_loc[item.char] = lc
5532            toloc = lc
5533            break
5534          end
5535        end
5536      end
5537    end
5538    % Now, take action, but treat composite chars in a different
5539    % fashion, because they 'inherit' the previous locale. Not yet
5540    % optimized.
5541    if not toloc and
5542        (item.char >= 0x0300 and item.char <= 0x036F) or
5543        (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
5544        (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
5545      toloc = toloc_save
5546    end
5547    if toloc and toloc > -1 then
5548      if Babel.locale_props[toloc].lg then
5549        item.lang = Babel.locale_props[toloc].lg
5550        node.set_attribute(item, LOCALE, toloc)
5551      end
5552      if Babel.locale_props[toloc]['/'..item.font] then
5553        item.font = Babel.locale_props[toloc]['/'..item.font]
5554      end
5555      toloc_save = toloc
5556    end
5557  elseif not inmath and item.id == 7 then
5558    item.replace = item.replace and Babel.locale_map(item.replace)
5559    item.pre     = item.pre and Babel.locale_map(item.pre)
5560    item.post    = item.post and Babel.locale_map(item.post)
5561  elseif item.id == node.id'math' then
5562    inmath = (item.subtype == 0)
5563  end
5564  end
5565  return head
5566 end
5567 }
```

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

```
5568 \newcommand\babelcharproperty[1]{%
5569   \count@=#1\relax
5570   \ifvmode
5571     \expandafter\bbl@chprop
5572   \else
5573     \bbl@error{\string\babelcharproperty\space can be used only in\\%
5574                vertical mode (preamble or between paragraphs)}%
5575                {See the manual for futher info}%
5576   \fi}
5577 \newcommand\bbl@chprop[3][\the\count@]{%
5578   \@tempcnta=#1\relax
5579   \bbl@ifunset{bbl@chprop@#2}%
5580     {\bbl@error{No property named '#2'. Allowed values are\\%
5581                direction (bc), mirror (bmg), and linebreak (lb)}%
5582                {See the manual for futher info}}%
5583     {}%
5584   \loop
5585     \bbl@cs{chprop@#2}{#3}%
5586   \ifnum\count@<\@tempcnta
```

183

```
5587    \advance\count@\@ne
5588  \repeat}
5589 \def\bbl@chprop@direction#1{%
5590  \directlua{
5591    Babel.characters[\the\count@] =  Babel.characters[\the\count@] or {}
5592    Babel.characters[\the\count@]['d'] = '#1'
5593  }}
5594 \let\bbl@chprop@bc\bbl@chprop@direction
5595 \def\bbl@chprop@mirror#1{%
5596  \directlua{
5597    Babel.characters[\the\count@] =  Babel.characters[\the\count@] or {}
5598    Babel.characters[\the\count@]['m'] = '\number#1'
5599  }}
5600 \let\bbl@chprop@bmg\bbl@chprop@mirror
5601 \def\bbl@chprop@linebreak#1{%
5602  \directlua{
5603    Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
5604    Babel.cjk_characters[\the\count@]['c'] = '#1'
5605  }}
5606 \let\bbl@chprop@lb\bbl@chprop@linebreak
5607 \def\bbl@chprop@locale#1{%
5608  \directlua{
5609    Babel.chr_to_loc = Babel.chr_to_loc or {}
5610    Babel.chr_to_loc[\the\count@] =
5611      \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@@#1}}\space
5612  }}
```

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.

```
5613 \directlua{
5614   Babel.nohyphenation = \the\l@nohyphenation
5615 }
```

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).

```
5616 \begingroup
5617 \catcode`\~=12
5618 \catcode`\%=12
5619 \catcode`\&=14
5620 \gdef\babelposthyphenation#1#2#3{&%
5621   \bbl@activateposthyphen
5622   \begingroup
5623     \def\babeltempa{\bbl@add@list\babeltempb}&%
5624     \let\babeltempb\@empty
5625     \def\bbl@tempa{#3}&% TODO. Ugly trick to preserve {}:
5626     \bbl@replace\bbl@tempa{,}{ ,}&%
5627     \expandafter\bbl@foreach\expandafter{\bbl@tempa}{&%
5628       \bbl@ifsamestring{##1}{remove}&%
5629         {\bbl@add@list\babeltempb{nil}}&%
5630         {\directlua{
5631           local rep = [=[##1]=]
5632           rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
```

184

```
5633          rep = rep:gsub('^%s*(insert)%s*,', 'insert = true, ')
5634          rep = rep:gsub(    '(no)%s*=%s*([^%s,]*)', Babel.capture_func)
5635          rep = rep:gsub(    '(pre)%s*=%s*([^%s,]*)', Babel.capture_func)
5636          rep = rep:gsub(    '(post)%s*=%s*([^%s,]*)', Babel.capture_func)
5637          rep = rep:gsub('(string)%s*=%s*([^%s,]*)', Babel.capture_func)
5638          tex.print([[\string\babeltempa{{]] .. rep .. [[}}]])
5639        }}}&%
5640    \directlua{
5641      local lbkr = Babel.linebreaking.replacements[1]
5642      local u = unicode.utf8
5643      local id = \the\csname l@#1\endcsname
5644      &% Convert pattern:
5645      local patt = string.gsub([==[#2]==], '%s', '')
5646      if not u.find(patt, '()', nil, true) then
5647        patt = '()' .. patt .. '()'
5648      end
5649      patt = string.gsub(patt, '%(%)%^', '^()')
5650      patt = string.gsub(patt, '%$%(%)', '()$')
5651      patt = u.gsub(patt, '{(.)}',
5652             function (n)
5653               return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
5654             end)
5655      patt = u.gsub(patt, '{(%x%x%x%x+)}',
5656             function (n)
5657               return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%1')
5658             end)
5659      lbkr[id] = lbkr[id] or {}
5660      table.insert(lbkr[id], { pattern = patt, replace = { \babeltempb } })
5661    }&%
5662  \endgroup}
5663 % TODO. Copypaste pattern.
5664 \gdef\babelprehyphenation#1#2#3{&%
5665    \bbl@activateprehyphen
5666    \begingroup
5667    \def\babeltempa{\bbl@add@list\babeltempb}&%
5668    \let\babeltempb\@empty
5669    \def\bbl@tempa{#3}&% TODO. Ugly trick to preserve {}:
5670    \bbl@replace\bbl@tempa{,}{ ,}&%
5671    \expandafter\bbl@foreach\expandafter{\bbl@tempa}{&%
5672      \bbl@ifsamestring{##1}{remove}&%
5673        {\bbl@add@list\babeltempb{nil}}&%
5674        {\directlua{
5675           local rep = [=[##1]=]
5676           rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
5677           rep = rep:gsub('^%s*(insert)%s*,', 'insert = true, ')
5678           rep = rep:gsub('(string)%s*=%s*([^%s,]*)', Babel.capture_func)
5679           rep = rep:gsub('(space)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
5680             'space = {' .. '%2, %3, %4' .. '}')
5681           rep = rep:gsub('(spacefactor)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
5682             'spacefactor = {' .. '%2, %3, %4' .. '}')
5683           rep = rep:gsub('(kashida)%s*=%s*([^%s,]*)', Babel.capture_kashida)
5684           tex.print([[\string\babeltempa{{]] .. rep .. [[}}]])
5685        }}}&%
5686    \directlua{
5687      local lbkr = Babel.linebreaking.replacements[0]
5688      local u = unicode.utf8
5689      local id = \the\csname bbl@id@@#1\endcsname
5690      &% Convert pattern:
5691      local patt = string.gsub([==[#2]==], '%s', '')
```

185

```
5692      local patt = string.gsub(patt, '|', ' ')
5693      if not u.find(patt, '()', nil, true) then
5694        patt = '()' .. patt .. '()'
5695      end
5696      &% patt = string.gsub(patt, '%(%)%^', '^()')
5697      &% patt = string.gsub(patt, '([^%%])%$%(%)', '%1()$')
5698      patt = u.gsub(patt, '{(.)}',
5699              function (n)
5700                return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
5701              end)
5702      patt = u.gsub(patt, '{(%x%x%x%x+)}',
5703              function (n)
5704                return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%%1')
5705              end)
5706      lbkr[id] = lbkr[id] or {}
5707      table.insert(lbkr[id], { pattern = patt, replace = { \babeltempb } })
5708    }&%
5709  \endgroup}
5710 \endgroup
5711 \def\bbl@activateposthyphen{%
5712  \let\bbl@activateposthyphen\relax
5713  \directlua{
5714    require('babel-transforms.lua')
5715    Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
5716  }}
5717 \def\bbl@activateprehyphen{%
5718  \let\bbl@activateprehyphen\relax
5719  \directlua{
5720    require('babel-transforms.lua')
5721    Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
5722  }}
```

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.

```
5723 \def\bbl@activate@preotf{%
5724  \let\bbl@activate@preotf\relax   % only once
5725  \directlua{
5726    Babel = Babel or {}
5727    %
5728    function Babel.pre_otfload_v(head)
5729      if Babel.numbers and Babel.digits_mapped then
5730        head = Babel.numbers(head)
5731      end
5732      if Babel.bidi_enabled then
5733        head = Babel.bidi(head, false, dir)
5734      end
5735      return head
5736    end
5737    %
5738    function Babel.pre_otfload_h(head, gc, sz, pt, dir)
5739      if Babel.numbers and Babel.digits_mapped then
5740        head = Babel.numbers(head)
5741      end
5742      if Babel.bidi_enabled then
5743        head = Babel.bidi(head, false, dir)
```

```
5744        end
5745        return head
5746      end
5747      %
5748      luatexbase.add_to_callback('pre_linebreak_filter',
5749        Babel.pre_otfload_v,
5750        'Babel.pre_otfload_v',
5751        luatexbase.priority_in_callback('pre_linebreak_filter',
5752          'luaotfload.node_processor') or nil)
5753      %
5754      luatexbase.add_to_callback('hpack_filter',
5755        Babel.pre_otfload_h,
5756        'Babel.pre_otfload_h',
5757        luatexbase.priority_in_callback('hpack_filter',
5758          'luaotfload.node_processor') or nil)
5759    }}
```

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=.

```
5760 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5761   \let\bbl@beforeforeign\leavevmode
5762   \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
5763   \RequirePackage{luatexbase}
5764   \bbl@activate@preotf
5765   \directlua{
5766     require('babel-data-bidi.lua')
5767     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
5768       require('babel-bidi-basic.lua')
5769     \or
5770       require('babel-bidi-basic-r.lua')
5771     \fi}
5772   % TODO - to locale_props, not as separate attribute
5773   \newattribute\bbl@attr@dir
5774   \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
5775   % TODO. I don't like it, hackish:
5776   \bbl@exp{\output{\bodydir\pagedir\the\output}}
5777   \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
5778 \fi\fi
5779 \chardef\bbl@thetextdir\z@
5780 \chardef\bbl@thepardir\z@
5781 \def\bbl@getluadir#1{%
5782   \directlua{
5783     if tex.#1dir == 'TLT' then
5784       tex.sprint('0')
5785     elseif tex.#1dir == 'TRT' then
5786       tex.sprint('1')
5787     end}}
5788 \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
5789   \ifcase#3\relax
5790     \ifcase\bbl@getluadir{#1}\relax\else
5791       #2 TLT\relax
5792     \fi
5793   \else
5794     \ifcase\bbl@getluadir{#1}\relax
5795       #2 TRT\relax
5796     \fi
5797   \fi}
5798 \def\bbl@textdir#1{%
```

```
5799    \bbl@setluadir{text}\textdir{#1}%
5800    \chardef\bbl@thetextdir#1\relax
5801    \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*3+#1}}
5802 \def\bbl@pardir#1{%
5803    \bbl@setluadir{par}\pardir{#1}%
5804    \chardef\bbl@thepardir#1\relax}
5805 \def\bbl@bodydir{\bbl@setluadir{body}\bodydir}
5806 \def\bbl@pagedir{\bbl@setluadir{page}\pagedir}
5807 \def\bbl@dirparastext{\pardir\the\textdir\relax}%    %%%%
5808 %
5809 \ifnum\bbl@bidimode>\z@
5810    \def\bbl@mathboxdir{%
5811      \ifcase\bbl@thetextdir\relax
5812        \everyhbox{\bbl@mathboxdir@aux L}%
5813      \else
5814        \everyhbox{\bbl@mathboxdir@aux R}%
5815      \fi}
5816    \def\bbl@mathboxdir@aux#1{%
5817      \@ifnextchar\egroup{}{\textdir T#1T\relax}}
5818    \frozen@everymath\expandafter{%
5819      \expandafter\bbl@mathboxdir\the\frozen@everymath}
5820    \frozen@everydisplay\expandafter{%
5821      \expandafter\bbl@mathboxdir\the\frozen@everydisplay}
5822 \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.

```
5823 \bbl@trace{Redefinitions for bidi layout}
5824 \ifx\@eqnnum\@undefined\else
5825    \ifx\bbl@attr@dir\@undefined\else
5826      \edef\@eqnnum{{%
5827        \unexpanded{\ifcase\bbl@attr@dir\else\bbl@textdir\@ne\fi}%
5828        \unexpanded\expandafter{\@eqnnum}}}
5829    \fi
5830 \fi
5831 \ifx\bbl@opt@layout\@nnil\endinput\fi  % if no layout
5832 \ifnum\bbl@bidimode>\z@
5833    \def\bbl@nextfake#1{%  non-local changes, use always inside a group!
5834      \bbl@exp{%
5835        \mathdir\the\bodydir
5836        #1%                 Once entered in math, set boxes to restore values
5837        \<ifmmode>%
5838          \everyvbox{%
5839            \the\everyvbox
5840            \bodydir\the\bodydir
5841            \mathdir\the\mathdir
```

```
5842            \everyhbox{\the\everyhbox}%
5843            \everyvbox{\the\everyvbox}}%
5844          \everyhbox{%
5845            \the\everyhbox
5846            \bodydir\the\bodydir
5847            \mathdir\the\mathdir
5848            \everyhbox{\the\everyhbox}%
5849            \everyvbox{\the\everyvbox}}%
5850        \<fi>}}%
5851    \def\@hangfrom#1{%
5852      \setbox\@tempboxa\hbox{{#1}}%
5853      \hangindent\wd\@tempboxa
5854      \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
5855        \shapemode\@ne
5856      \fi
5857      \noindent\box\@tempboxa}
5858  \fi
5859  \IfBabelLayout{tabular}
5860    {\let\bbl@OL@@tabular\@tabular
5861     \bbl@replace\@tabular{$}{\bbl@nextfake$}%
5862     \let\bbl@NL@@tabular\@tabular
5863     \AtBeginDocument{%
5864       \ifx\bbl@NL@@tabular\@tabular\else
5865         \bbl@replace\@tabular{$}{\bbl@nextfake$}%
5866         \let\bbl@NL@@tabular\@tabular
5867       \fi}}
5868     {}
5869  \IfBabelLayout{lists}
5870    {\let\bbl@OL@list\list
5871     \bbl@sreplace\list{\parshape}{\bbl@listparshape}%
5872     \let\bbl@NL@list\list
5873     \def\bbl@listparshape#1#2#3{%
5874       \parshape #1 #2 #3 %
5875       \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
5876         \shapemode\tw@
5877       \fi}}
5878     {}
5879  \IfBabelLayout{graphics}
5880    {\let\bbl@pictresetdir\relax
5881     \def\bbl@pictsetdir#1{%
5882       \ifcase\bbl@thetextdir
5883         \let\bbl@pictresetdir\relax
5884       \else
5885         \ifcase#1\bodydir TLT  % Remember this sets the inner boxes
5886           \or\textdir TLT
5887           \else\bodydir TLT \textdir TLT
5888         \fi
5889         % \(text|par)dir required in pgf:
5890         \def\bbl@pictresetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}%
5891       \fi}%
5892     \ifx\AddToHook\@undefined\else
5893       \AddToHook{env/picture/begin}{\bbl@pictsetdir\tw@}%
5894       \directlua{
5895         Babel.get_picture_dir = true
5896         Babel.picture_has_bidi = 0
5897         function Babel.picture_dir (head)
5898           if not Babel.get_picture_dir then return head end
5899           for item in node.traverse(head) do
5900             if item.id == node.id'glyph' then
```

189

```
5901              local itemchar = item.char
5902              % TODO. Copypaste pattern from Babel.bidi (-r)
5903              local chardata = Babel.characters[itemchar]
5904              local dir = chardata and chardata.d or nil
5905              if not dir then
5906                for nn, et in ipairs(Babel.ranges) do
5907                  if itemchar < et[1] then
5908                    break
5909                  elseif itemchar <= et[2] then
5910                    dir = et[3]
5911                    break
5912                  end
5913                end
5914              end
5915              if dir and (dir == 'al' or dir == 'r') then
5916                Babel.picture_has_bidi = 1
5917              end
5918            end
5919          end
5920          return head
5921        end
5922        luatexbase.add_to_callback("hpack_filter", Babel.picture_dir,
5923          "Babel.picture_dir")
5924    }%
5925  \AtBeginDocument{%
5926    \long\def\put(#1,#2)#3{%
5927      \@killglue
5928      % Try:
5929      \ifx\bbl@pictresetdir\relax
5930        \def\bbl@tempc{0}%
5931      \else
5932        \directlua{
5933          Babel.get_picture_dir = true
5934          Babel.picture_has_bidi = 0
5935        }%
5936        \setbox\z@\hb@xt@\z@{%
5937          \@defaultunitsset\@tempdimc{#1}\unitlength
5938          \kern\@tempdimc
5939          #3\hss}%
5940        \edef\bbl@tempc{\directlua{tex.print(Babel.picture_has_bidi)}}%
5941      \fi
5942      % Do:
5943      \@defaultunitsset\@tempdimc{#2}\unitlength
5944      \raise\@tempdimc\hb@xt@\z@{%
5945        \@defaultunitsset\@tempdimc{#1}\unitlength
5946        \kern\@tempdimc
5947        {\ifnum\bbl@tempc>\z@\bbl@pictresetdir\fi#3}\hss}%
5948      \ignorespaces}%
5949    \MakeRobust\put}%
5950  \fi
5951  \AtBeginDocument
5952    {\ifx\tikz@atbegin@node\@undefined\else
5953      \ifx\AddToHook\@undefined\else % TODO. Still tentative.
5954        \AddToHook{env/pgfpicture/begin}{\bbl@pictsetdir\@ne}%
5955        % \AddToHook{env/tikzpicture/begin}{\bbl@pictsetdir\z@}%
5956        \bbl@add\pgfinterruptpicture{\bbl@pictresetdir}%
5957      \fi
5958      \let\bbl@OL@pgfpicture\pgfpicture
5959      \bbl@sreplace\pgfpicture{\pgfpicturetrue}%
```

```
5960          {\bbl@pictsetdir\z@\pgfpicturetrue}%
5961        \bbl@add\pgfsys@beginpicture{\bbl@pictsetdir\z@}%
5962        \bbl@add\tikz@atbegin@node{\bbl@pictresetdir}%
5963        \bbl@sreplace\tikz{\begingroup}%
5964          {\begingroup\bbl@pictsetdir\tw@}%
5965      \fi
5966      \ifx\AddToHook\@undefined\else
5967        \AddToHook{env/tcolorbox/begin}{\bbl@pictsetdir\@ne}%
5968      \fi
5969      }}
5970    {}
```

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`.

```
5971 \IfBabelLayout{counters}%
5972    {\let\bbl@OL@@textsuperscript\@textsuperscript
5973     \bbl@sreplace\@textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
5974     \let\bbl@latinarabic=\@arabic
5975     \let\bbl@OL@@arabic\@arabic
5976     \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
5977     \@ifpackagewith{babel}{bidi=default}%
5978       {\let\bbl@asciiroman=\@roman
5979        \let\bbl@OL@@roman\@roman
5980        \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciiroman#1}}}%
5981        \let\bbl@asciiRoman=\@Roman
5982        \let\bbl@OL@@roman\@Roman
5983        \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}%
5984        \let\bbl@OL@labelenumii\labelenumii
5985        \def\labelenumii(){\theenumii(}%
5986        \let\bbl@OL@p@enumiii\p@enumiii
5987        \def\p@enumiii{\p@enumii)\theenumii(}}{}}{}%
5988 ⟨⟨Footnote changes⟩⟩
5989 \IfBabelLayout{footnotes}%
5990    {\let\bbl@OL@footnote\footnote
5991     \BabelFootnote\footnote\languagename{}{}%
5992     \BabelFootnote\localfootnote\languagename{}{}%
5993     \BabelFootnote\mainfootnote{}{}{}}
5994    {}
```

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.

```
5995 \IfBabelLayout{extras}%
5996    {\let\bbl@OL@underline\underline
5997     \bbl@sreplace\underline{$\@@underline}{\bbl@nextfake$\@@underline}%
5998     \let\bbl@OL@LaTeX2e\LaTeX2e
5999     \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
6000       \if b\expandafter\@car\f@series\@nil\boldmath\fi
6001       \babelsublr{%
6002         \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}}}
6003    {}
6004 ⟨/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,

191

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.

```
6005 ⟨∗transforms⟩
6006 Babel.linebreaking.replacements = {}
6007 Babel.linebreaking.replacements[0] = {}  -- pre
6008 Babel.linebreaking.replacements[1] = {}  -- post
6009
6010 -- Discretionaries contain strings as nodes
6011 function Babel.str_to_nodes(fn, matches, base)
6012   local n, head, last
6013   if fn == nil then return nil end
6014   for s in string.utfvalues(fn(matches)) do
6015     if base.id == 7 then
6016       base = base.replace
6017     end
6018     n = node.copy(base)
6019     n.char    = s
6020     if not head then
6021       head = n
6022     else
6023       last.next = n
6024     end
6025     last = n
6026   end
6027   return head
6028 end
6029
6030 Babel.fetch_subtext = {}
6031
6032 Babel.ignore_pre_char = function(node)
6033   return (node.lang == Babel.nohyphenation)
6034 end
6035
6036 -- Merging both functions doesn't seen feasible, because there are too
6037 -- many differences.
6038 Babel.fetch_subtext[0] = function(head)
6039   local word_string = ''
6040   local word_nodes = {}
6041   local lang
6042   local item = head
6043   local inmath = false
6044
6045   while item do
6046
6047     if item.id == 11 then
6048       inmath = (item.subtype == 0)
6049     end
6050
6051     if inmath then
6052       -- pass
6053
6054     elseif item.id == 29 then
```

```
6055      local locale = node.get_attribute(item, Babel.attr_locale)
6056
6057      if lang == locale or lang == nil then
6058        lang = lang or locale
6059        if Babel.ignore_pre_char(item) then
6060          word_string = word_string .. Babel.us_char
6061        else
6062          word_string = word_string .. unicode.utf8.char(item.char)
6063        end
6064        word_nodes[#word_nodes+1] = item
6065      else
6066        break
6067      end
6068
6069    elseif item.id == 12 and item.subtype == 13 then
6070      word_string = word_string .. ' '
6071      word_nodes[#word_nodes+1] = item
6072
6073    -- Ignore leading unrecognized nodes, too.
6074    elseif word_string ~= '' then
6075      word_string = word_string .. Babel.us_char
6076      word_nodes[#word_nodes+1] = item  -- Will be ignored
6077    end
6078
6079    item = item.next
6080  end
6081
6082  -- Here and above we remove some trailing chars but not the
6083  -- corresponding nodes. But they aren't accessed.
6084  if word_string:sub(-1) == ' ' then
6085    word_string = word_string:sub(1,-2)
6086  end
6087  word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
6088  return word_string, word_nodes, item, lang
6089 end
6090
6091 Babel.fetch_subtext[1] = function(head)
6092  local word_string = ''
6093  local word_nodes = {}
6094  local lang
6095  local item = head
6096  local inmath = false
6097
6098  while item do
6099
6100    if item.id == 11 then
6101      inmath = (item.subtype == 0)
6102    end
6103
6104    if inmath then
6105      -- pass
6106
6107    elseif item.id == 29 then
6108      if item.lang == lang or lang == nil then
6109        if (item.char ~= 124) and (item.char ~= 61) then -- not =, not |
6110          lang = lang or item.lang
6111          word_string = word_string .. unicode.utf8.char(item.char)
6112          word_nodes[#word_nodes+1] = item
6113        end
```

```
6114        else
6115          break
6116        end
6117
6118      elseif item.id == 7 and item.subtype == 2 then
6119        word_string = word_string .. '='
6120        word_nodes[#word_nodes+1] = item
6121
6122      elseif item.id == 7 and item.subtype == 3 then
6123        word_string = word_string .. '|'
6124        word_nodes[#word_nodes+1] = item
6125
6126      -- (1) Go to next word if nothing was found, and (2) implicitly
6127      -- remove leading USs.
6128      elseif word_string == '' then
6129        -- pass
6130
6131      -- This is the responsible for splitting by words.
6132      elseif (item.id == 12 and item.subtype == 13) then
6133        break
6134
6135      else
6136        word_string = word_string .. Babel.us_char
6137        word_nodes[#word_nodes+1] = item  -- Will be ignored
6138      end
6139
6140      item = item.next
6141    end
6142
6143    word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
6144    return word_string, word_nodes, item, lang
6145 end
6146
6147 function Babel.pre_hyphenate_replace(head)
6148    Babel.hyphenate_replace(head, 0)
6149 end
6150
6151 function Babel.post_hyphenate_replace(head)
6152    Babel.hyphenate_replace(head, 1)
6153 end
6154
6155 Babel.us_char = string.char(31)
6156
6157 function Babel.hyphenate_replace(head, mode)
6158    local u = unicode.utf8
6159    local lbkr = Babel.linebreaking.replacements[mode]
6160
6161    local word_head = head
6162
6163    while true do  -- for each subtext block
6164
6165      local w, w_nodes, nw, lang = Babel.fetch_subtext[mode](word_head)
6166
6167      if Babel.debug then
6168        print()
6169        print((mode == 0) and '@@@@<' or '@@@@>', w)
6170      end
6171
6172      if nw == nil and w == '' then break end
```

```
6173
6174    if not lang then goto next end
6175    if not lbkr[lang] then goto next end
6176
6177    -- For each saved (pre|post)hyphenation. TODO. Reconsider how
6178    -- loops are nested.
6179    for k=1, #lbkr[lang] do
6180      local p = lbkr[lang][k].pattern
6181      local r = lbkr[lang][k].replace
6182
6183      if Babel.debug then
6184        print('*****', p, mode)
6185      end
6186
6187      -- This variable is set in some cases below to the first *byte*
6188      -- after the match, either as found by u.match (faster) or the
6189      -- computed position based on sc if w has changed.
6190      local last_match = 0
6191      local step = 0
6192
6193      -- For every match.
6194      while true do
6195        if Babel.debug then
6196          print('=====')
6197        end
6198        local new  -- used when inserting and removing nodes
6199
6200        local matches = { u.match(w, p, last_match) }
6201
6202        if #matches < 2 then break end
6203
6204        -- Get and remove empty captures (with ()'s, which return a
6205        -- number with the position), and keep actual captures
6206        -- (from (...)), if any, in matches.
6207        local first = table.remove(matches, 1)
6208        local last  = table.remove(matches, #matches)
6209        -- Non re-fetched substrings may contain \31, which separates
6210        -- subsubstrings.
6211        if string.find(w:sub(first, last-1), Babel.us_char) then break end
6212
6213        local save_last = last -- with A()BC()D, points to D
6214
6215        -- Fix offsets, from bytes to unicode. Explained above.
6216        first = u.len(w:sub(1, first-1)) + 1
6217        last  = u.len(w:sub(1, last-1)) -- now last points to C
6218
6219        -- This loop stores in n small table the nodes
6220        -- corresponding to the pattern. Used by 'data' to provide a
6221        -- predictable behavior with 'insert' (now w_nodes is modified on
6222        -- the fly), and also access to 'remove'd nodes.
6223        local sc = first-1           -- Used below, too
6224        local data_nodes = {}
6225
6226        for q = 1, last-first+1 do
6227          data_nodes[q] = w_nodes[sc+q]
6228        end
6229
6230        -- This loop traverses the matched substring and takes the
6231        -- corresponding action stored in the replacement list.
```

```
6232          -- sc = the position in substr nodes / string
6233          -- rc = the replacement table index
6234          local rc = 0
6235
6236          while rc < last-first+1 do -- for each replacement
6237            if Babel.debug then
6238              print('.....', rc + 1)
6239            end
6240            sc = sc + 1
6241            rc = rc + 1
6242
6243            if Babel.debug then
6244              Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6245              local ss = ''
6246              for itt in node.traverse(head) do
6247               if itt.id == 29 then
6248                 ss = ss .. unicode.utf8.char(itt.char)
6249               else
6250                 ss = ss .. '{' .. itt.id .. '}'
6251               end
6252              end
6253              print('*****************', ss)
6254
6255            end
6256
6257            local crep = r[rc]
6258            local item = w_nodes[sc]
6259            local item_base = item
6260            local placeholder = Babel.us_char
6261            local d
6262
6263            if crep and crep.data then
6264              item_base = data_nodes[crep.data]
6265            end
6266
6267            if crep then
6268              step = crep.step or 0
6269            end
6270
6271            if crep and next(crep) == nil then -- = {}
6272              last_match = save_last    -- Optimization
6273              goto next
6274
6275            elseif crep == nil or crep.remove then
6276              node.remove(head, item)
6277              table.remove(w_nodes, sc)
6278              w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
6279              sc = sc - 1  -- Nothing has been inserted.
6280              last_match = utf8.offset(w, sc+1+step)
6281              goto next
6282
6283            elseif crep and crep.kashida then -- Experimental
6284              node.set_attribute(item,
6285                  Babel.attr_kashida,
6286                  crep.kashida)
6287              last_match = utf8.offset(w, sc+1+step)
6288              goto next
6289
6290            elseif crep and crep.string then
```

```
6291              local str = crep.string(matches)
6292            if str == '' then  -- Gather with nil
6293              node.remove(head, item)
6294              table.remove(w_nodes, sc)
6295              w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
6296              sc = sc - 1  -- Nothing has been inserted.
6297            else
6298              local loop_first = true
6299              for s in string.utfvalues(str) do
6300                d = node.copy(item_base)
6301                d.char = s
6302                if loop_first then
6303                  loop_first = false
6304                  head, new = node.insert_before(head, item, d)
6305                  if sc == 1 then
6306                    word_head = head
6307                  end
6308                  w_nodes[sc] = d
6309                  w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc+1)
6310                else
6311                  sc = sc + 1
6312                  head, new = node.insert_before(head, item, d)
6313                  table.insert(w_nodes, sc, new)
6314                  w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc)
6315                end
6316                if Babel.debug then
6317                  print('.....', 'str')
6318                  Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6319                end
6320              end  -- for
6321              node.remove(head, item)
6322            end  -- if ''
6323            last_match = utf8.offset(w, sc+1+step)
6324            goto next
6325
6326          elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
6327            d = node.new(7, 0)   -- (disc, discretionary)
6328            d.pre     = Babel.str_to_nodes(crep.pre, matches, item_base)
6329            d.post    = Babel.str_to_nodes(crep.post, matches, item_base)
6330            d.replace = Babel.str_to_nodes(crep.no, matches, item_base)
6331            d.attr = item_base.attr
6332            if crep.pre == nil then  -- TeXbook p96
6333              d.penalty = crep.penalty or tex.hyphenpenalty
6334            else
6335              d.penalty = crep.penalty or tex.exhyphenpenalty
6336            end
6337            placeholder = '|'
6338            head, new = node.insert_before(head, item, d)
6339
6340          elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
6341            -- ERROR
6342
6343          elseif crep and crep.penalty then
6344            d = node.new(14, 0)   -- (penalty, userpenalty)
6345            d.attr = item_base.attr
6346            d.penalty = crep.penalty
6347            head, new = node.insert_before(head, item, d)
6348
6349          elseif crep and crep.space then
```

```
6350              -- 655360 = 10 pt = 10 * 65536 sp
6351              d = node.new(12, 13)      -- (glue, spaceskip)
6352              local quad = font.getfont(item_base.font).size or 655360
6353              node.setglue(d, crep.space[1] * quad,
6354                            crep.space[2] * quad,
6355                            crep.space[3] * quad)
6356              if mode == 0 then
6357                placeholder = ' '
6358              end
6359              head, new = node.insert_before(head, item, d)
6360
6361          elseif crep and crep.spacefactor then
6362              d = node.new(12, 13)      -- (glue, spaceskip)
6363              local base_font = font.getfont(item_base.font)
6364              node.setglue(d,
6365                crep.spacefactor[1] * base_font.parameters['space'],
6366                crep.spacefactor[2] * base_font.parameters['space_stretch'],
6367                crep.spacefactor[3] * base_font.parameters['space_shrink'])
6368              if mode == 0 then
6369                placeholder = ' '
6370              end
6371              head, new = node.insert_before(head, item, d)
6372
6373          elseif mode == 0 and crep and crep.space then
6374              -- ERROR
6375
6376          end  -- ie replacement cases
6377
6378          -- Shared by disc, space and penalty.
6379          if sc == 1 then
6380            word_head = head
6381          end
6382          if crep.insert then
6383            w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc)
6384            table.insert(w_nodes, sc, new)
6385            last = last + 1
6386          else
6387            w_nodes[sc] = d
6388            node.remove(head, item)
6389            w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1)
6390          end
6391
6392          last_match = utf8.offset(w, sc+1+step)
6393
6394          ::next::
6395
6396        end  -- for each replacement
6397
6398        if Babel.debug then
6399            print('.....', '/')
6400            Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6401        end
6402
6403      end  -- for match
6404
6405    end  -- for patterns
6406
6407    ::next::
6408    word_head = nw
```

```
6409   end  -- for substring
6410   return head
6411 end
6412
6413 -- This table stores capture maps, numbered consecutively
6414 Babel.capture_maps = {}
6415
6416 -- The following functions belong to the next macro
6417 function Babel.capture_func(key, cap)
6418   local ret = "[[" .. cap:gsub('{([0-9])}', "]]..m[%1]..[[") .. "]]"
6419   local cnt
6420   local u = unicode.utf8
6421   ret, cnt = ret:gsub('{([0-9])|([^|]+)|(.-)}', Babel.capture_func_map)
6422   if cnt == 0 then
6423     ret = u.gsub(ret, '{(%x%x%x%x+)}',
6424           function (n)
6425             return u.char(tonumber(n, 16))
6426           end)
6427   end
6428   ret = ret:gsub("%[%[%]%]%.%.", '')
6429   ret = ret:gsub("%.%.%[%[%]%]", '')
6430   return key .. [[=function(m) return ]] .. ret .. [[ end]]
6431 end
6432
6433 function Babel.capt_map(from, mapno)
6434   return Babel.capture_maps[mapno][from] or from
6435 end
6436
6437 -- Handle the {n|abc|ABC} syntax in captures
6438 function Babel.capture_func_map(capno, from, to)
6439   local u = unicode.utf8
6440   from = u.gsub(from, '{(%x%x%x%x+)}',
6441         function (n)
6442           return u.char(tonumber(n, 16))
6443         end)
6444   to = u.gsub(to, '{(%x%x%x%x+)}',
6445         function (n)
6446           return u.char(tonumber(n, 16))
6447         end)
6448   local froms = {}
6449   for s in string.utfcharacters(from) do
6450     table.insert(froms, s)
6451   end
6452   local cnt = 1
6453   table.insert(Babel.capture_maps, {})
6454   local mlen = table.getn(Babel.capture_maps)
6455   for s in string.utfcharacters(to) do
6456     Babel.capture_maps[mlen][froms[cnt]] = s
6457     cnt = cnt + 1
6458   end
6459   return "]]..Babel.capt_map(m[" .. capno .. "]," ..
6460         (mlen) .. ")..)" .. "[["
6461 end
6462
6463 -- Create/Extend reversed sorted list of kashida weights:
6464 function Babel.capture_kashida(key, wt)
6465   wt = tonumber(wt)
6466   if Babel.kashida_wts then
6467     for p, q in ipairs(Babel.kashida_wts) do
```

```
6468      if wt  == q then
6469        break
6470      elseif wt > q then
6471        table.insert(Babel.kashida_wts, p, wt)
6472        break
6473      elseif table.getn(Babel.kashida_wts) == p then
6474        table.insert(Babel.kashida_wts, wt)
6475      end
6476    end
6477  else
6478    Babel.kashida_wts = { wt }
6479  end
6480  return 'kashida = ' .. wt
6481 end
6482 ⟨/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>).

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.

```
6483 ⟨*basic-r⟩
6484 Babel = Babel or {}
6485
6486 Babel.bidi_enabled = true
```

200

```
6487
6488 require('babel-data-bidi.lua')
6489
6490 local characters = Babel.characters
6491 local ranges = Babel.ranges
6492
6493 local DIR = node.id("dir")
6494
6495 local function dir_mark(head, from, to, outer)
6496   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
6497   local d = node.new(DIR)
6498   d.dir = '+' .. dir
6499   node.insert_before(head, from, d)
6500   d = node.new(DIR)
6501   d.dir = '-' .. dir
6502   node.insert_after(head, to, d)
6503 end
6504
6505 function Babel.bidi(head, ispar)
6506   local first_n, last_n          -- first and last char with nums
6507   local last_es                  -- an auxiliary 'last' used with nums
6508   local first_d, last_d          -- first and last char in L/R block
6509   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):

```
6510   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
6511   local strong_lr = (strong == 'l') and 'l' or 'r'
6512   local outer = strong
6513
6514   local new_dir = false
6515   local first_dir = false
6516   local inmath = false
6517
6518   local last_lr
6519
6520   local type_n = ''
6521
6522   for item in node.traverse(head) do
6523
6524     -- three cases: glyph, dir, otherwise
6525     if item.id == node.id'glyph'
6526       or (item.id == 7 and item.subtype == 2) then
6527
6528       local itemchar
6529       if item.id == 7 and item.subtype == 2 then
6530         itemchar = item.replace.char
6531       else
6532         itemchar = item.char
6533       end
6534       local chardata = characters[itemchar]
6535       dir = chardata and chardata.d or nil
6536       if not dir then
6537         for nn, et in ipairs(ranges) do
6538           if itemchar < et[1] then
6539             break
6540           elseif itemchar <= et[2] then
6541             dir = et[3]
```

```
6542              break
6543            end
6544          end
6545        end
6546        dir = dir or 'l'
6547        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).

```
6548        if new_dir then
6549          attr_dir = 0
6550          for at in node.traverse(item.attr) do
6551            if at.number == Babel.attr_dir then
6552              attr_dir = at.value % 3
6553            end
6554          end
6555          if attr_dir == 1 then
6556            strong = 'r'
6557          elseif attr_dir == 2 then
6558            strong = 'al'
6559          else
6560            strong = 'l'
6561          end
6562          strong_lr = (strong == 'l') and 'l' or 'r'
6563          outer = strong_lr
6564          new_dir = false
6565        end
6566
6567        if dir == 'nsm' then dir = strong end              -- W1
```

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

```
6568        dir_real = dir              -- We need dir_real to set strong below
6569        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:

```
6570        if strong == 'al' then
6571          if dir == 'en' then dir = 'an' end                    -- W2
6572          if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
6573          strong_lr = 'r'                                      -- W3
6574        end
```

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

```
6575      elseif item.id == node.id'dir' and not inmath then
6576        new_dir = true
6577        dir = nil
6578      elseif item.id == node.id'math' then
6579        inmath = (item.subtype == 0)
6580      else
6581        dir = nil          -- Not a char
6582      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>.

```
6583     if dir == 'en' or dir == 'an' or dir == 'et' then
6584       if dir ~= 'et' then
6585         type_n = dir
6586       end
6587       first_n = first_n or item
6588       last_n = last_es or item
6589       last_es = nil
6590     elseif dir == 'es' and last_n then -- W3+W6
6591       last_es = item
6592     elseif dir == 'cs' then              -- it's right - do nothing
6593     elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
6594       if strong_lr == 'r' and type_n ~= '' then
6595         dir_mark(head, first_n, last_n, 'r')
6596       elseif strong_lr == 'l' and first_d and type_n == 'an' then
6597         dir_mark(head, first_n, last_n, 'r')
6598         dir_mark(head, first_d, last_d, outer)
6599         first_d, last_d = nil, nil
6600       elseif strong_lr == 'l' and type_n ~= '' then
6601         last_d = last_n
6602       end
6603       type_n = ''
6604       first_n, last_n = nil, nil
6605     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:

```
6606     if dir == 'l' or dir == 'r' then
6607       if dir ~= outer then
6608         first_d = first_d or item
6609         last_d = item
6610       elseif first_d and dir ~= strong_lr then
6611         dir_mark(head, first_d, last_d, outer)
6612         first_d, last_d = nil, nil
6613       end
6614     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.

```
6615     if dir and not last_lr and dir ~= 'l' and outer == 'r' then
6616       item.char = characters[item.char] and
6617                   characters[item.char].m or item.char
6618     elseif (dir or new_dir) and last_lr ~= item then
6619       local mir = outer .. strong_lr .. (dir or outer)
6620       if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
6621         for ch in node.traverse(node.next(last_lr)) do
6622           if ch == item then break end
6623           if ch.id == node.id'glyph' and characters[ch.char] then
6624             ch.char = characters[ch.char].m or ch.char
6625           end
6626         end
6627       end
6628     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).

```
6629    if dir == 'l' or dir == 'r' then
6630      last_lr = item
6631      strong = dir_real              -- Don't search back - best save now
6632      strong_lr = (strong == 'l') and 'l' or 'r'
6633    elseif new_dir then
6634      last_lr = nil
6635    end
6636  end
```

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

```
6637  if last_lr and outer == 'r' then
6638    for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
6639      if characters[ch.char] then
6640        ch.char = characters[ch.char].m or ch.char
6641      end
6642    end
6643  end
6644  if first_n then
6645    dir_mark(head, first_n, last_n, outer)
6646  end
6647  if first_d then
6648    dir_mark(head, first_d, last_d, outer)
6649  end
```

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

```
6650    return node.prev(head) or head
6651 end
6652 ⟨/basic-r⟩
```

And here the Lua code for bidi=basic:

```
6653 ⟨*basic⟩
6654 Babel = Babel or {}
6655
6656 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
6657
6658 Babel.fontmap = Babel.fontmap or {}
6659 Babel.fontmap[0] = {}      -- l
6660 Babel.fontmap[1] = {}      -- r
6661 Babel.fontmap[2] = {}      -- al/an
6662
6663 Babel.bidi_enabled = true
6664 Babel.mirroring_enabled = true
6665
6666 require('babel-data-bidi.lua')
6667
6668 local characters = Babel.characters
6669 local ranges = Babel.ranges
6670
6671 local DIR = node.id('dir')
6672 local GLYPH = node.id('glyph')
6673
6674 local function insert_implicit(head, state, outer)
6675   local new_state = state
6676   if state.sim and state.eim and state.sim ~= state.eim then
6677     dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
6678     local d = node.new(DIR)
6679     d.dir = '+' .. dir
6680     node.insert_before(head, state.sim, d)
```

```lua
6681      local d = node.new(DIR)
6682      d.dir = '-' .. dir
6683      node.insert_after(head, state.eim, d)
6684    end
6685    new_state.sim, new_state.eim = nil, nil
6686    return head, new_state
6687  end
6688
6689  local function insert_numeric(head, state)
6690    local new
6691    local new_state = state
6692    if state.san and state.ean and state.san ~= state.ean then
6693      local d = node.new(DIR)
6694      d.dir = '+TLT'
6695      _, new = node.insert_before(head, state.san, d)
6696      if state.san == state.sim then state.sim = new end
6697      local d = node.new(DIR)
6698      d.dir = '-TLT'
6699      _, new = node.insert_after(head, state.ean, d)
6700      if state.ean == state.eim then state.eim = new end
6701    end
6702    new_state.san, new_state.ean = nil, nil
6703    return head, new_state
6704  end
6705
6706  -- TODO - \hbox with an explicit dir can lead to wrong results
6707  -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
6708  -- was s made to improve the situation, but the problem is the 3-dir
6709  -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
6710  -- well.
6711
6712  function Babel.bidi(head, ispar, hdir)
6713    local d    -- d is used mainly for computations in a loop
6714    local prev_d = ''
6715    local new_d = false
6716
6717    local nodes = {}
6718    local outer_first = nil
6719    local inmath = false
6720
6721    local glue_d = nil
6722    local glue_i = nil
6723
6724    local has_en = false
6725    local first_et = nil
6726
6727    local ATDIR = Babel.attr_dir
6728
6729    local save_outer
6730    local temp = node.get_attribute(head, ATDIR)
6731    if temp then
6732      temp = temp % 3
6733      save_outer = (temp == 0 and 'l') or
6734                   (temp == 1 and 'r') or
6735                   (temp == 2 and 'al')
6736    elseif ispar then           -- Or error? Shouldn't happen
6737      save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
6738    else                        -- Or error? Shouldn't happen
6739      save_outer = ('TRT' == hdir) and 'r' or 'l'
```

```
6740    end
6741      -- when the callback is called, we are just _after_ the box,
6742      -- and the textdir is that of the surrounding text
6743    -- if not ispar and hdir ~= tex.textdir then
6744    --   save_outer = ('TRT' == hdir) and 'r' or 'l'
6745    -- end
6746    local outer = save_outer
6747    local last = outer
6748    -- 'al' is only taken into account in the first, current loop
6749    if save_outer == 'al' then save_outer = 'r' end
6750
6751    local fontmap = Babel.fontmap
6752
6753    for item in node.traverse(head) do
6754
6755      -- In what follows, #node is the last (previous) node, because the
6756      -- current one is not added until we start processing the neutrals.
6757
6758      -- three cases: glyph, dir, otherwise
6759      if item.id == GLYPH
6760          or (item.id == 7 and item.subtype == 2) then
6761
6762        local d_font = nil
6763        local item_r
6764        if item.id == 7 and item.subtype == 2 then
6765          item_r = item.replace    -- automatic discs have just 1 glyph
6766        else
6767          item_r = item
6768        end
6769        local chardata = characters[item_r.char]
6770        d = chardata and chardata.d or nil
6771        if not d or d == 'nsm' then
6772          for nn, et in ipairs(ranges) do
6773            if item_r.char < et[1] then
6774              break
6775            elseif item_r.char <= et[2] then
6776              if not d then d = et[3]
6777              elseif d == 'nsm' then d_font = et[3]
6778              end
6779              break
6780            end
6781          end
6782        end
6783        d = d or 'l'
6784
6785        -- A short 'pause' in bidi for mapfont
6786        d_font = d_font or d
6787        d_font = (d_font == 'l' and 0) or
6788                 (d_font == 'nsm' and 0) or
6789                 (d_font == 'r' and 1) or
6790                 (d_font == 'al' and 2) or
6791                 (d_font == 'an' and 2) or nil
6792        if d_font and fontmap and fontmap[d_font][item_r.font] then
6793          item_r.font = fontmap[d_font][item_r.font]
6794        end
6795
6796        if new_d then
6797          table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
6798          if inmath then
```

```
6799          attr_d = 0
6800        else
6801          attr_d = node.get_attribute(item, ATDIR)
6802          attr_d = attr_d % 3
6803        end
6804        if attr_d == 1 then
6805          outer_first = 'r'
6806          last = 'r'
6807        elseif attr_d == 2 then
6808          outer_first = 'r'
6809          last = 'al'
6810        else
6811          outer_first = 'l'
6812          last = 'l'
6813        end
6814        outer = last
6815        has_en = false
6816        first_et = nil
6817        new_d = false
6818      end
6819
6820      if glue_d then
6821        if (d == 'l' and 'l' or 'r') ~= glue_d then
6822          table.insert(nodes, {glue_i, 'on', nil})
6823        end
6824        glue_d = nil
6825        glue_i = nil
6826      end
6827
6828    elseif item.id == DIR then
6829      d = nil
6830      new_d = true
6831
6832    elseif item.id == node.id'glue' and item.subtype == 13 then
6833      glue_d = d
6834      glue_i = item
6835      d = nil
6836
6837    elseif item.id == node.id'math' then
6838      inmath = (item.subtype == 0)
6839
6840    else
6841      d = nil
6842    end
6843
6844    -- AL <= EN/ET/ES      -- W2 + W3 + W6
6845    if last == 'al' and d == 'en' then
6846      d = 'an'              -- W3
6847    elseif last == 'al' and (d == 'et' or d == 'es') then
6848      d = 'on'             -- W6
6849    end
6850
6851    -- EN + CS/ES + EN      -- W4
6852    if d == 'en' and #nodes >= 2 then
6853      if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
6854          and nodes[#nodes-1][2] == 'en' then
6855        nodes[#nodes][2] = 'en'
6856      end
6857    end
```

```
6858
6859    -- AN + CS + AN        -- W4 too, because uax9 mixes both cases
6860    if d == 'an' and #nodes >= 2 then
6861      if (nodes[#nodes][2] == 'cs')
6862          and nodes[#nodes-1][2] == 'an' then
6863        nodes[#nodes][2] = 'an'
6864      end
6865    end
6866
6867    -- ET/EN                -- W5 + W7->l / W6->on
6868    if d == 'et' then
6869      first_et = first_et or (#nodes + 1)
6870    elseif d == 'en' then
6871      has_en = true
6872      first_et = first_et or (#nodes + 1)
6873    elseif first_et then      -- d may be nil here !
6874      if has_en then
6875        if last == 'l' then
6876          temp = 'l'    -- W7
6877        else
6878          temp = 'en'   -- W5
6879        end
6880      else
6881        temp = 'on'     -- W6
6882      end
6883      for e = first_et, #nodes do
6884        if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
6885      end
6886      first_et = nil
6887      has_en = false
6888    end
6889
6890    -- Force mathdir in math if ON (currently works as expected only
6891    -- with 'l')
6892    if inmath and d == 'on' then
6893      d = ('TRT' == tex.mathdir) and 'r' or 'l'
6894    end
6895
6896    if d then
6897      if d == 'al' then
6898        d = 'r'
6899        last = 'al'
6900      elseif d == 'l' or d == 'r' then
6901        last = d
6902      end
6903      prev_d = d
6904      table.insert(nodes, {item, d, outer_first})
6905    end
6906
6907    outer_first = nil
6908
6909  end
6910
6911  -- TODO -- repeated here in case EN/ET is the last node. Find a
6912  -- better way of doing things:
6913  if first_et then        -- dir may be nil here !
6914    if has_en then
6915      if last == 'l' then
6916        temp = 'l'    -- W7
```

```
6917        else
6918          temp = 'en'    -- W5
6919        end
6920      else
6921        temp = 'on'      -- W6
6922      end
6923      for e = first_et, #nodes do
6924        if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
6925      end
6926    end
6927
6928    -- dummy node, to close things
6929    table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
6930
6931    --------------  NEUTRAL -----------------
6932
6933    outer = save_outer
6934    last = outer
6935
6936    local first_on = nil
6937
6938    for q = 1, #nodes do
6939      local item
6940
6941      local outer_first = nodes[q][3]
6942      outer = outer_first or outer
6943      last = outer_first or last
6944
6945      local d = nodes[q][2]
6946      if d == 'an' or d == 'en' then d = 'r' end
6947      if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
6948
6949      if d == 'on' then
6950        first_on = first_on or q
6951      elseif first_on then
6952        if last == d then
6953          temp = d
6954        else
6955          temp = outer
6956        end
6957        for r = first_on, q - 1 do
6958          nodes[r][2] = temp
6959          item = nodes[r][1]    -- MIRRORING
6960          if Babel.mirroring_enabled and item.id == GLYPH
6961              and temp == 'r' and characters[item.char] then
6962            local font_mode = font.fonts[item.font].properties.mode
6963            if font_mode ~= 'harf' and font_mode ~= 'plug' then
6964              item.char = characters[item.char].m or item.char
6965            end
6966          end
6967        end
6968        first_on = nil
6969      end
6970
6971      if d == 'r' or d == 'l' then last = d end
6972    end
6973
6974    --------------  IMPLICIT, REORDER ----------------
6975
```

```lua
6976    outer = save_outer
6977    last = outer
6978
6979    local state = {}
6980    state.has_r = false
6981
6982    for q = 1, #nodes do
6983
6984      local item = nodes[q][1]
6985
6986      outer = nodes[q][3] or outer
6987
6988      local d = nodes[q][2]
6989
6990      if d == 'nsm' then d = last end            -- W1
6991      if d == 'en' then d = 'an' end
6992      local isdir = (d == 'r' or d == 'l')
6993
6994      if outer == 'l' and d == 'an' then
6995        state.san = state.san or item
6996        state.ean = item
6997      elseif state.san then
6998        head, state = insert_numeric(head, state)
6999      end
7000
7001      if outer == 'l' then
7002        if d == 'an' or d == 'r' then      -- im -> implicit
7003          if d == 'r' then state.has_r = true end
7004          state.sim = state.sim or item
7005          state.eim = item
7006        elseif d == 'l' and state.sim and state.has_r then
7007          head, state = insert_implicit(head, state, outer)
7008        elseif d == 'l' then
7009          state.sim, state.eim, state.has_r = nil, nil, false
7010        end
7011      else
7012        if d == 'an' or d == 'l' then
7013          if nodes[q][3] then -- nil except after an explicit dir
7014            state.sim = item  -- so we move sim 'inside' the group
7015          else
7016            state.sim = state.sim or item
7017          end
7018          state.eim = item
7019        elseif d == 'r' and state.sim then
7020          head, state = insert_implicit(head, state, outer)
7021        elseif d == 'r' then
7022          state.sim, state.eim = nil, nil
7023        end
7024      end
7025
7026      if isdir then
7027        last = d             -- Don't search back - best save now
7028      elseif d == 'on' and state.san  then
7029        state.san = state.san or item
7030        state.ean = item
7031      end
7032
7033    end
7034
```

```
7035  return node.prev(head) or head
7036 end
7037 ⟨/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.

```
7038 ⟨*nil⟩
7039 \ProvidesLanguage{nil}[⟨⟨date⟩⟩ ⟨⟨version⟩⟩ Nil language]
7040 \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.

```
7041 \ifx\l@nil\@undefined
7042   \newlanguage\l@nil
7043   \@namedef{bbl@hyphendata@\the\l@nil}{{}{}}% Remove warning
7044   \let\bbl@elt\relax
7045   \edef\bbl@languages{%  Add it to the list of languages
7046     \bbl@languages\bbl@elt{nil}{\the\l@nil}{}{}}
7047 \fi
```

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

```
7048 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}
```

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

\captionnil
\datenil
```
7049 \let\captionsnil\@empty
7050 \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.

```
7051 \ldf@finish{nil}
7052 ⟨/nil⟩
```

211

16   Support for Plain T_EX (`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 T_EX-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 iniT_EX, 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 iniT_EX sees, we need to set some category codes just to be able to change the definition of \input.

```
7053 ⟨*bplain | blplain⟩
7054 \catcode`\{=1 % left brace is begin-group character
7055 \catcode`\}=2 % right brace is end-group character
7056 \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).

```
7057 \openin 0 hyphen.cfg
7058 \ifeof0
7059 \else
7060   \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.

```
7061   \def\input #1 {%
7062     \let\input\a
7063     \a hyphen.cfg
7064     \let\a\undefined
7065   }
7066 \fi
7067 ⟨/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.

```
7068 ⟨bplain⟩\a plain.tex
7069 ⟨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.

```
7070 ⟨bplain⟩\def\fmtname{babel-plain}
7071 ⟨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 L^AT_EX features

The file babel.def expects some definitions made in the L^AT_EX 2_ε style file. So, in Plain we must provide at least some predefined values as well some tools to set them (even if not all options are available). There are no package options, and therefore and alternative mechanism is provided. For

the moment, only `\babeloptionstrings` and `\babeloptionmath` are provided, which can be defined before loading babel. `\BabelModifiers` can be set too (but not sure it works).

```
7072 ⟨⟨∗Emulate LaTeX⟩⟩ ≡
7073 \def\@empty{}
7074 \def\loadlocalcfg#1{%
7075   \openin0#1.cfg
7076   \ifeof0
7077     \closein0
7078   \else
7079     \closein0
7080     {\immediate\write16{***********************************}%
7081      \immediate\write16{* Local config file #1.cfg used}%
7082      \immediate\write16{*}%
7083      }
7084     \input #1.cfg\relax
7085   \fi
7086   \@endofldf}
```

16.3   General tools

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

```
7087 \long\def\@firstofone#1{#1}
7088 \long\def\@firstoftwo#1#2{#1}
7089 \long\def\@secondoftwo#1#2{#2}
7090 \def\@nnil{\@nil}
7091 \def\@gobbletwo#1#2{}
7092 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
7093 \def\@star@or@long#1{%
7094   \@ifstar
7095   {\let\l@ngrel@x\relax#1}%
7096   {\let\l@ngrel@x\long#1}}
7097 \let\l@ngrel@x\relax
7098 \def\@car#1#2\@nil{#1}
7099 \def\@cdr#1#2\@nil{#2}
7100 \let\@typeset@protect\relax
7101 \let\protected@edef\edef
7102 \long\def\@gobble#1{}
7103 \edef\@backslashchar{\expandafter\@gobble\string\\}
7104 \def\strip@prefix#1>{}
7105 \def\g@addto@macro#1#2{{%
7106     \toks@\expandafter{#1#2}%
7107     \xdef#1{\the\toks@}}}
7108 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
7109 \def\@nameuse#1{\csname #1\endcsname}
7110 \def\@ifundefined#1{%
7111   \expandafter\ifx\csname#1\endcsname\relax
7112     \expandafter\@firstoftwo
7113   \else
7114     \expandafter\@secondoftwo
7115   \fi}
7116 \def\@expandtwoargs#1#2#3{%
7117   \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
7118 \def\zap@space#1 #2{%
7119   #1%
7120   \ifx#2\@empty\else\expandafter\zap@space\fi
7121   #2}
7122 \let\bbl@trace\@gobble
7123 \def\bbl@error#1#2{%
```

```
7124    \begingroup
7125      \newlinechar=`\^^J
7126      \def\\{^^J(babel) }%
7127      \errhelp{#2}\errmessage{\\#1}%
7128    \endgroup}
7129 \def\bbl@warning#1{%
7130    \begingroup
7131      \newlinechar=`\^^J
7132      \def\\{^^J(babel) }%
7133      \message{\\#1}%
7134    \endgroup}
7135 \let\bbl@infowarn\bbl@warning
7136 \def\bbl@info#1{%
7137    \begingroup
7138      \newlinechar=`\^^J
7139      \def\\{^^J}%
7140      \wlog{#1}%
7141    \endgroup}
```

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

```
7142 \ifx\@preamblecmds\@undefined
7143    \def\@preamblecmds{}
7144 \fi
7145 \def\@onlypreamble#1{%
7146    \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
7147      \@preamblecmds\do#1}}
7148 \@onlypreamble\@onlypreamble
```

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

```
7149 \def\begindocument{%
7150    \@begindocumenthook
7151    \global\let\@begindocumenthook\@undefined
7152    \def\do##1{\global\let##1\@undefined}%
7153    \@preamblecmds
7154    \global\let\do\noexpand}
7155 \ifx\@begindocumenthook\@undefined
7156    \def\@begindocumenthook{}
7157 \fi
7158 \@onlypreamble\@begindocumenthook
7159 \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.

```
7160 \def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
7161 \@onlypreamble\AtEndOfPackage
7162 \def\@endofldf{}
7163 \@onlypreamble\@endofldf
7164 \let\bbl@afterlang\@empty
7165 \chardef\bbl@opt@hyphenmap\z@
```

LaTeX needs to be able to switch off writing to its auxiliary files; plain doesn't have them by default. There is a trick to hide some conditional commands from the outer \ifx. The same trick is applied below.

```
7166 \catcode`\&=\z@
7167 \ifx&if@filesw\@undefined
7168    \expandafter\let\csname if@filesw\expandafter\endcsname
7169      \csname iffalse\endcsname
7170 \fi
7171 \catcode`\&=4
```

Mimick LaTeX's commands to define control sequences.

```
7172 \def\newcommand{\@star@or@long\new@command}
7173 \def\new@command#1{%
7174   \@testopt{\@newcommand#1}0}
7175 \def\@newcommand#1[#2]{%
7176   \@ifnextchar [{\@xargdef#1[#2]}%
7177                 {\@argdef#1[#2]}}
7178 \long\def\@argdef#1[#2]#3{%
7179   \@yargdef#1\@ne{#2}{#3}}
7180 \long\def\@xargdef#1[#2][#3]#4{%
7181   \expandafter\def\expandafter#1\expandafter{%
7182     \expandafter\@protected@testopt\expandafter #1%
7183     \csname\string#1\expandafter\endcsname{#3}}%
7184   \expandafter\@yargdef \csname\string#1\endcsname
7185   \tw@{#2}{#4}}
7186 \long\def\@yargdef#1#2#3{%
7187   \@tempcnta#3\relax
7188   \advance \@tempcnta \@ne
7189   \let\@hash@\relax
7190   \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
7191   \@tempcntb #2%
7192   \@whilenum\@tempcntb <\@tempcnta
7193   \do{%
7194     \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
7195     \advance\@tempcntb \@ne}%
7196   \let\@hash@##%
7197   \l@ngrel@x\expandafter\def\expandafter#1\reserved@a}
7198 \def\providecommand{\@star@or@long\provide@command}
7199 \def\provide@command#1{%
7200   \begingroup
7201     \escapechar\m@ne\xdef\@gtempa{{\string#1}}%
7202   \endgroup
7203   \expandafter\@ifundefined\@gtempa
7204     {\def\reserved@a{\new@command#1}}%
7205     {\let\reserved@a\relax
7206      \def\reserved@a{\new@command\reserved@a}}%
7207   \reserved@a}%

7208 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
7209 \def\declare@robustcommand#1{%
7210   \edef\reserved@a{\string#1}%
7211   \def\reserved@b{#1}%
7212   \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
7213   \edef#1{%
7214     \ifx\reserved@a\reserved@b
7215       \noexpand\x@protect
7216       \noexpand#1%
7217     \fi
7218     \noexpand\protect
7219     \expandafter\noexpand\csname
7220       \expandafter\@gobble\string#1 \endcsname
7221   }%
7222   \expandafter\new@command\csname
7223     \expandafter\@gobble\string#1 \endcsname
7224 }
7225 \def\x@protect#1{%
7226   \ifx\protect\@typeset@protect\else
7227     \@x@protect#1%
7228   \fi
```

215

```
7229 }
7230 \catcode`\&=\z@  % Trick to hide conditionals
7231   \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.

```
7232   \def\bbl@tempa{\csname newif\endcsname&ifin@}
7233 \catcode`\&=4
7234 \ifx\in@\@undefined
7235   \def\in@#1#2{%
7236     \def\in@@##1#1##2##3\in@@{%
7237       \ifx\in@@##2\in@false\else\in@true\fi}%
7238     \in@@#2#1\in@\in@@}
7239 \else
7240   \let\bbl@tempa\@empty
7241 \fi
7242 \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).

```
7243 \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.

```
7244 \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.

```
7245 \ifx\@tempcnta\@undefined
7246   \csname newcount\endcsname\@tempcnta\relax
7247 \fi
7248 \ifx\@tempcntb\@undefined
7249   \csname newcount\endcsname\@tempcntb\relax
7250 \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).

```
7251 \ifx\bye\@undefined
7252   \advance\count10 by -2\relax
7253 \fi
7254 \ifx\@ifnextchar\@undefined
7255   \def\@ifnextchar#1#2#3{%
7256     \let\reserved@d=#1%
7257     \def\reserved@a{#2}\def\reserved@b{#3}%
7258     \futurelet\@let@token\@ifnch}
7259   \def\@ifnch{%
7260     \ifx\@let@token\@sptoken
7261       \let\reserved@c\@xifnch
7262     \else
7263       \ifx\@let@token\reserved@d
7264         \let\reserved@c\reserved@a
7265       \else
7266         \let\reserved@c\reserved@b
7267       \fi
7268     \fi
```

```
7269      \reserved@c}
7270   \def\:{\let\@sptoken= } \:  % this makes \@sptoken a space token
7271   \def\:{\@xifnch} \expandafter\def\: {\futurelet\@let@token\@ifnch}
7272 \fi
7273 \def\@testopt#1#2{%
7274   \@ifnextchar[{#1}{#1[#2]}}
7275 \def\@protected@testopt#1{%
7276   \ifx\protect\@typeset@protect
7277     \expandafter\@testopt
7278   \else
7279     \@x@protect#1%
7280   \fi}
7281 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
7282      #2\relax}\fi}
7283 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
7284        \else\expandafter\@gobble\fi{#1}}
```

16.4   Encoding related macros

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

```
7285 \def\DeclareTextCommand{%
7286   \@dec@text@cmd\providecommand
7287 }
7288 \def\ProvideTextCommand{%
7289   \@dec@text@cmd\providecommand
7290 }
7291 \def\DeclareTextSymbol#1#2#3{%
7292   \@dec@text@cmd\chardef#1{#2}#3\relax
7293 }
7294 \def\@dec@text@cmd#1#2#3{%
7295   \expandafter\def\expandafter#2%
7296      \expandafter{%
7297        \csname#3-cmd\expandafter\endcsname
7298        \expandafter#2%
7299        \csname#3\string#2\endcsname
7300      }%
7301 %   \let\@ifdefinable\@rc@ifdefinable
7302   \expandafter#1\csname#3\string#2\endcsname
7303 }
7304 \def\@current@cmd#1{%
7305   \ifx\protect\@typeset@protect\else
7306      \noexpand#1\expandafter\@gobble
7307   \fi
7308 }
7309 \def\@changed@cmd#1#2{%
7310   \ifx\protect\@typeset@protect
7311      \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
7312        \expandafter\ifx\csname ?\string#1\endcsname\relax
7313          \expandafter\def\csname ?\string#1\endcsname{%
7314            \@changed@x@err{#1}%
7315          }%
7316        \fi
7317        \global\expandafter\let
7318          \csname\cf@encoding \string#1\expandafter\endcsname
7319          \csname ?\string#1\endcsname
7320      \fi
7321      \csname\cf@encoding\string#1%
7322        \expandafter\endcsname
```

```
7323    \else
7324      \noexpand#1%
7325    \fi
7326 }
7327 \def\@changed@x@err#1{%
7328    \errhelp{Your command will be ignored, type <return> to proceed}%
7329    \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
7330 \def\DeclareTextCommandDefault#1{%
7331    \DeclareTextCommand#1?%
7332 }
7333 \def\ProvideTextCommandDefault#1{%
7334    \ProvideTextCommand#1?%
7335 }
7336 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
7337 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
7338 \def\DeclareTextAccent#1#2#3{%
7339   \DeclareTextCommand#1{#2}[1]{\accent#3 ##1}
7340 }
7341 \def\DeclareTextCompositeCommand#1#2#3#4{%
7342    \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
7343    \edef\reserved@b{\string##1}%
7344    \edef\reserved@c{%
7345      \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
7346    \ifx\reserved@b\reserved@c
7347      \expandafter\expandafter\expandafter\ifx
7348        \expandafter\@car\reserved@a\relax\relax\@nil
7349        \@text@composite
7350      \else
7351        \edef\reserved@b##1{%
7352          \def\expandafter\noexpand
7353            \csname#2\string#1\endcsname####1{%
7354            \noexpand\@text@composite
7355              \expandafter\noexpand\csname#2\string#1\endcsname
7356              ####1\noexpand\@empty\noexpand\@text@composite
7357              {##1}%
7358          }%
7359        }%
7360        \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
7361      \fi
7362      \expandafter\def\csname\expandafter\string\csname
7363        #2\endcsname\string#1-\string#3\endcsname{#4}
7364    \else
7365      \errhelp{Your command will be ignored, type <return> to proceed}%
7366      \errmessage{\string\DeclareTextCompositeCommand\space used on
7367        inappropriate command \protect#1}
7368    \fi
7369 }
7370 \def\@text@composite#1#2#3\@text@composite{%
7371    \expandafter\@text@composite@x
7372      \csname\string#1-\string#2\endcsname
7373 }
7374 \def\@text@composite@x#1#2{%
7375    \ifx#1\relax
7376      #2%
7377    \else
7378      #1%
7379    \fi
7380 }
7381 %
```

218

```
7382 \def\@strip@args#1:#2-#3\@strip@args{#2}
7383 \def\DeclareTextComposite#1#2#3#4{%
7384     \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
7385     \bgroup
7386         \lccode`\@=#4%
7387         \lowercase{%
7388     \egroup
7389         \reserved@a @%
7390     }%
7391 }
7392 %
7393 \def\UseTextSymbol#1#2{#2}
7394 \def\UseTextAccent#1#2#3{}
7395 \def\@use@text@encoding#1{}
7396 \def\DeclareTextSymbolDefault#1#2{%
7397     \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
7398 }
7399 \def\DeclareTextAccentDefault#1#2{%
7400     \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
7401 }
7402 \def\cf@encoding{OT1}
```

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

```
7403 \DeclareTextAccent{\"}{OT1}{127}
7404 \DeclareTextAccent{\'}{OT1}{19}
7405 \DeclareTextAccent{\^}{OT1}{94}
7406 \DeclareTextAccent{\`}{OT1}{18}
7407 \DeclareTextAccent{\~}{OT1}{126}
```

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

```
7408 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
7409 \DeclareTextSymbol{\textquotedblright}{OT1}{`\"}
7410 \DeclareTextSymbol{\textquoteleft}{OT1}{`\`}
7411 \DeclareTextSymbol{\textquoteright}{OT1}{`\'}
7412 \DeclareTextSymbol{\i}{OT1}{16}
7413 \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.

```
7414 \ifx\scriptsize\@undefined
7415   \let\scriptsize\sevenrm
7416 \fi
```

And a few more "dummy" definitions.

```
7417 \def\languagename{english}%
7418 \let\bbl@opt@shorthands\@nnil
7419 \def\bbl@ifshorthand#1#2#3{#2}%
7420 \let\bbl@language@opts\@empty
7421 \ifx\babeloptionstrings\@undefined
7422   \let\bbl@opt@strings\@nnil
7423 \else
7424   \let\bbl@opt@strings\babeloptionstrings
7425 \fi
7426 \def\BabelStringsDefault{generic}
7427 \def\bbl@tempa{normal}
7428 \ifx\babeloptionmath\bbl@tempa
7429   \def\bbl@mathnormal{\noexpand\textormath}
7430 \fi
```

```
7431 \def\AfterBabelLanguage#1#2{}
7432 \ifx\BabelModifiers\@undefined\let\BabelModifiers\relax\fi
7433 \let\bbl@afterlang\relax
7434 \def\bbl@opt@safe{BR}
7435 \ifx\@uclclist\@undefined\let\@uclclist\@empty\fi
7436 \ifx\bbl@trace\@undefined\def\bbl@trace#1{}\fi
7437 \expandafter\newif\csname ifbbl@single\endcsname
7438 \chardef\bbl@bidimode\z@
7439 ⟨⟨/Emulate LaTeX⟩⟩
```

A proxy file:

```
7440 ⟨*plain⟩
7441 \input babel.def
7442 ⟨/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).