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 repository. 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.

\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).
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

10

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. 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:

```
  ! 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.

`\useshorthands` *{⟨*char*⟩}

The command `\useshorthands` initiates the definition of user-defined shorthand sequences. It has one argument, the character that starts these personal shorthands.

New 3.9a   User shorthands are not always alive, as they may be deactivated by languages (for example, if you use " for your user shorthands and switch from german to french, they stop working). Therefore, a starred version `\useshorthands*{⟨`char`⟩}` is provided, which makes sure shorthands are always activated.

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

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

The command `\defineshorthand` takes two arguments: the first is a one- or two-character shorthand sequence, and the second is the code the shorthand should expand to.

New 3.9a   An optional argument allows to (re)define language and system shorthands (some languages do not activate shorthands, so you may want to add `\languageshorthands{⟨`lang`⟩}` to the corresponding `\extras`⟨*lang*⟩, as explained below). By default, user shorthands are (re)defined.

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

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

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

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

---

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

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

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

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

\languageshorthands {⟨*language*⟩}

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

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

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

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

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

\babelshorthand {⟨*shorthand*⟩}

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

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

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

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

Languages with no shorthands Croatian, English (any variety), Indonesian, Hebrew, Interlingua, Irish, Lower Sorbian, Malaysian, North Sami, Romanian, Scottish, Welsh

---

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

[6]Thanks to Enrico Gregorio

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.

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.

15

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.

---

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

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.

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

---

[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.

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]{1∩ 1ɰ 1ə 1ŋ 1ɲ 1ๆ} % 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]	bo	Tibetan[u]
agq	Aghem	brx	Bodo
ak	Akan	bs-Cyrl	Bosnian
am	Amharic[ul]	bs-Latn	Bosnian[ul]
ar	Arabic[ul]	bs	Bosnian[ul]
ar-DZ	Arabic[ul]	ca	Catalan[ul]
ar-MA	Arabic[ul]	ce	Chechen
ar-SY	Arabic[ul]	cgg	Chiga
as	Assamese	chr	Cherokee
asa	Asu	ckb	Central Kurdish
ast	Asturian[ul]	cop	Coptic
az-Cyrl	Azerbaijani	cs	Czech[ul]
az-Latn	Azerbaijani	cu	Church Slavic
az	Azerbaijani[ul]	cu-Cyrs	Church Slavic
bas	Basaa	cu-Glag	Church Slavic
be	Belarusian[ul]	cy	Welsh[ul]
bem	Bemba	da	Danish[ul]
bez	Bena	dav	Taita
bg	Bulgarian[ul]	de-AT	German[ul]
bm	Bambara	de-CH	German[ul]
bn	Bangla[ul]	de	German[ul]

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
hsb	Upper Sorbian[ul]	mr	Marathi[ul]
hu	Hungarian[ul]	ms-BN	Malay[l]
hy	Armenian[u]	ms-SG	Malay[l]
ia	Interlingua[ul]	ms	Malay[ul]
id	Indonesian[ul]	mt	Maltese
ig	Igbo	mua	Mundang

my	Burmese
mzn	Mazanderani
naq	Nama
nb	Norwegian Bokmål[ul]
nd	North Ndebele
ne	Nepali
nl	Dutch[ul]
nmg	Kwasio
nn	Norwegian Nynorsk[ul]
nnh	Ngiemboon
nus	Nuer
nyn	Nyankole
om	Oromo
or	Odia
os	Ossetic
pa-Arab	Punjabi
pa-Guru	Punjabi
pa	Punjabi
pl	Polish[ul]
pms	Piedmontese[ul]
ps	Pashto
pt-BR	Portuguese[ul]
pt-PT	Portuguese[ul]
pt	Portuguese[ul]
qu	Quechua
rm	Romansh[ul]
rn	Rundi
ro	Romanian[ul]
rof	Rombo
ru	Russian[ul]
rw	Kinyarwanda
rwk	Rwa
sa-Beng	Sanskrit
sa-Deva	Sanskrit
sa-Gujr	Sanskrit
sa-Knda	Sanskrit
sa-Mlym	Sanskrit
sa-Telu	Sanskrit
sa	Sanskrit
sah	Sakha
saq	Samburu
sbp	Sangu
se	Northern Sami[ul]
seh	Sena
ses	Koyraboro Senni
sg	Sango
shi-Latn	Tachelhit
shi-Tfng	Tachelhit
shi	Tachelhit
si	Sinhala
sk	Slovak[ul]
sl	Slovenian[ul]
smn	Inari Sami
sn	Shona
so	Somali
sq	Albanian[ul]
sr-Cyrl-BA	Serbian[ul]
sr-Cyrl-ME	Serbian[ul]
sr-Cyrl-XK	Serbian[ul]
sr-Cyrl	Serbian[ul]
sr-Latn-BA	Serbian[ul]
sr-Latn-ME	Serbian[ul]
sr-Latn-XK	Serbian[ul]
sr-Latn	Serbian[ul]
sr	Serbian[ul]
sv	Swedish[ul]
sw	Swahili
ta	Tamil[u]
te	Telugu[ul]
teo	Teso
th	Thai[ul]
ti	Tigrinya
tk	Turkmen[ul]
to	Tongan
tr	Turkish[ul]
twq	Tasawaq
tzm	Central Atlas Tamazight
ug	Uyghur
uk	Ukrainian[ul]
ur	Urdu[ul]
uz-Arab	Uzbek
uz-Cyrl	Uzbek
uz-Latn	Uzbek
uz	Uzbek
vai-Latn	Vai
vai-Vaii	Vai
vai	Vai
vi	Vietnamese[ul]
vun	Vunjo
wae	Walser
xog	Soga
yav	Yangben
yi	Yiddish
yo	Yoruba
yue	Cantonese
zgh	Standard Moroccan Tamazight
zh-Hans-HK	Chinese
zh-Hans-MO	Chinese
zh-Hans-SG	Chinese
zh-Hans	Chinese
zh-Hant-HK	Chinese
zh-Hant-MO	Chinese
zh-Hant	Chinese
zh	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	cantonese
akan	catalan
albanian	centralatlastamazight
american	centralkurdish
amharic	chechen
ancientgreek	cherokee
arabic	chiga
arabic-algeria	chinese-hans-hk
arabic-DZ	chinese-hans-mo
arabic-morocco	chinese-hans-sg
arabic-MA	chinese-hans
arabic-syria	chinese-hant-hk
arabic-SY	chinese-hant-mo
armenian	chinese-hant
assamese	chinese-simplified-hongkongsarchina
asturian	chinese-simplified-macausarchina
asu	chinese-simplified-singapore
australian	chinese-simplified
austrian	chinese-traditional-hongkongsarchina
azerbaijani-cyrillic	chinese-traditional-macausarchina
azerbaijani-cyrl	chinese-traditional
azerbaijani-latin	chinese
azerbaijani-latn	churchslavic
azerbaijani	churchslavic-cyrs
bafia	churchslavic-oldcyrillic[12]
bambara	churchsslavic-glag
basaa	churchsslavic-glagolitic
basque	colognian
belarusian	cornish
bemba	croatian
bena	czech
bengali	danish
bodo	duala
bosnian-cyrillic	dutch
bosnian-cyrl	dzongkha
bosnian-latin	embu
bosnian-latn	english-au
bosnian	english-australia
brazilian	english-ca
breton	english-canada
british	english-gb
bulgarian	english-newzealand
burmese	english-nz
canadian	english-unitedkingdom

---

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

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
ukrainian
uppersorbian
urdu

usenglish	vai-vaii
usorbian	vai
uyghur	vietnam
uzbek-arab	vietnamese
uzbek-arabic	vunjo
uzbek-cyrillic	walser
uzbek-cyrl	welsh
uzbek-latin	westernfrisian
uzbek-latn	yangben
uzbek	yiddish
vai-latin	yoruba
vai-latn	zarma
vai-vai	zulu afrikaans

Modifying and adding values to `ini` **files**

New 3.39   There is a way to modify the values of `ini` files when they get loaded with
`\babelprovide` and `import`. To set, say, `digits.native` in the `numbers` section, use
something like `numbers/digits.native=abcdefghij`. Keys may be added, too. Without
`import` you may modify the identification keys.

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

1.14   Selecting fonts

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

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

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

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

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

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

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

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

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

---

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

26

```
\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:

```
\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:

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

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

NOTE You may load fontspec explicitly. For example:

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

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

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

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

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

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

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

This is *not* and 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* and 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}%
}
```

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.

<dl>
<dt>captions=</dt>
<dd>

⟨*language-tag*⟩

Loads only the strings. For example:

```
\babelprovide[captions=hu]{hungarian}
```
</dd>

<dt>hyphenrules=</dt>
<dd>

⟨*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.
</dd>

<dt>main</dt>
<dd>

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}
```
</dd>

<dt>script=</dt>
<dd>

⟨*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.
</dd>
</dl>

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).

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.

\localeid

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

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

1.20 Hyphenation and line breaking

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

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

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

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

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

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

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

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

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

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

New 3.9a Sets hyphenation exceptions for the languages given or, without the optional argument, for *all* languages (eg, proper nouns or common loan words, and of course monolingual documents). Language exceptions take precedence over global ones.
It can be used only in the preamble, and exceptions are set when the language is first selected, thus taking into account changes of \lccodes's done in \extras⟨lang⟩ as well as the language-specific encoding (not set in the preamble by default). Multiple \babelhyphenation's are allowed. For example:

36

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

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

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

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

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

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

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

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

1.21 Transforms

Transforms (only luatex) provide a way to process the text on the typesetting level in several language-dependent ways, like non-standard hyphenation, special line breaking rules, script to script conversion, spacing conventions and so on.[15]

---

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

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

It currently embraces \babelprehyphenation and \babelposthyphenation.

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.
	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.
Norsk	doubleletter.hyphen	Hyphenates the doble-letter groups *bb*, *dd*, *ff*, *gg*, *ll*, *mm*, *nn*, *pp*, *rr*, *ss*, *tt* as *bb-b*, *dd-d*, etc.
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*⟩}

With luatex it is now 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. No rules are currently provided by default, but they can be defined as shown in the following example, where {1} is the first captured char (between ( ) in the pattern):

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

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

This feature is activated with the first \babelposthyphenation or \babelprehyphenation. See the babel site for a more detailed description and some examples. It also describes a few additional replacement types (string, penalty).

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

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

\babelprehyphenation   {⟨*locale-name*⟩}{⟨*lua-pattern*⟩}{⟨*replacement*⟩}

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

It handles glyphs and spaces.

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

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

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

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

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

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

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

1.22 Selection based on BCP 47 tags

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

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

Here is a minimal example:

```
\documentclass{article}

\usepackage[danish]{babel}

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

\begin{document}

Chapter in Danish: \chaptername.

\selectlanguage{de-AT}

\localedate{2020}{1}{30}

\end{document}
```

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

The behaviour is adjusted with `\babeladjust` with the following parameters:

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

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

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

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

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

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

1.23 Selecting scripts

Currently babel provides no standard interface to select scripts, because they are best selected with either `\fontencoding` (low-level) or a language name (high-level). Even the Latin script may require different encodings (ie, sets of glyphs) depending on the language, and therefore such a switch would be in a sense incomplete.[16]
Some languages sharing the same script define macros to switch it (eg, `\textcyrillic`), but be aware they may also set the language to a certain default. Even the babel core defined `\textlatin`, but is was somewhat buggy because in some cases it messed up encodings and fonts (for example, if the main Latin encoding was LY1), and therefore it has been deprecated.[17]

`\ensureascii` {⟨*text*⟩}

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

1.24 Selecting directions

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

WARNING The current code for **text** in luatex should be considered essentially stable, but, of course, it is not bug-free and there can be improvements in the future, because setting bidi text has many subtleties (see for example <https://www.w3.org/TR/html-bidi/>). A basic stable version for other engines must wait. This applies to text; there is a basic support for **graphical** elements, including

---

[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.

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:

42

```
\documentclass{book}

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

\babelprovide[onchar=ids fonts]{arabic}

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

\begin{document}

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

\end{document}
```

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

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

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

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

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

New 3.16 *To be expanded.* Selects which layout elements are adapted in bidi documents, including some text elements (except with options loading the bidi package, which provides its own mechanism to control these elements). You may use several options with a 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]

---

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

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

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

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

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

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

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

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

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

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

EXAMPLE Typically, in an Arabic document you would need:

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

\babelsublr {⟨*lr-text*⟩}

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

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

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

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

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

\BabelPatchSection   {⟨*section-name*⟩}

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

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

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

New 3.17   Something like:

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

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

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

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

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

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

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

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

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

1.25   Language attributes

\languageattribute

This is a user-level command, to be used in the preamble of a document (after \usepackage[...]{babel}), that declares which attributes are to be used for a given language. It takes two arguments: the first is the name of the language; the second, a (list of) attribute(s) to be used. Attributes must be set in the preamble and only once – they

cannot be turned on and off. The command checks whether the language is known in this document and whether the attribute(s) are known for this language.

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

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

1.26 Hooks

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

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

The same name can be applied to several events. Hooks 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.

46

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.

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:

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

```
\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`. For example, you can set `\babeladjust{bidi.text=off}` if you are using an alternative algorithm or with large sections not requiring it. With luahbtex you may need `bidi.mirroring=off`. 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 `\foreinglanguage`, 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 no known workaround.

---

[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.

- 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.
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

---

[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.

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:

```
% 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
```

---

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

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

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

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

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

A typical error when using babel is the following:

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

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

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

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

The following assumptions are made:

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

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

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

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

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

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

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.

- 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.

---

[26]But not removed, for backward compatibility.

The following page provides a starting point for `ldf` files:
http://www.texnia.com/incubator.html. See also
https://github.com/latex3/babel/blob/master/news-guides/guides/list-of-locale-templates.md.
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,

---

[27]This mechanism was introduced by Bernd Raichle.

56

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 below). In other words, the old way of defining/switching strings still works and it's used by default.

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

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

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

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

A "selector" is a name to be used as value in package option strings, optionally followed by extra info about the encodings to be used. The name unicode must be used for xetex and luatex (the key strings has also other two special values: generic and encoded). If a string is set several times (because several blocks are read), the first one takes precedence (ie, it works much like \providecommand).

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

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

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

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

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

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

\EndBabelCommands
```

A real example is:

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

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

---

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

```
\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.

\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.

---

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

\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 and error with a conditional.

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

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

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

Part II
Source code

babel is being developed incrementally, which means parts of the code are under development and therefore incomplete. Only documented features are considered complete. In other words, use babel only as documented (except, of course, if you want to explore and test them – you can post suggestions about multilingual issues to kadingira@tug.org on http://tug.org/mailman/listinfo/kadingira).

5 Identification and loading of required files

Code documentation is still under revision.

The following description is no longer valid, because switch and plain have been merged into babel.def.

The babel package after unpacking consists of the following files:

switch.def defines macros to set and switch languages.
babel.def defines the rest of macros. It has tow parts: a generic one and a second one only for LaTeX.
babel.sty is the LaTeX package, which set options and load language styles.
plain.def defines some LaTeX macros required by `babel.def` and provides a few tools for Plain.
hyphen.cfg is the file to be used when generating the formats to load hyphenation patterns.

The babel installer extends docstrip with a few "pseudo-guards" to set "variables" used at installation time. They are used with <@name@> at the appropiated places in the source code and shown below with ⟨⟨*name*⟩⟩. That brings a little bit of literate programming.

6 `locale` directory

A required component of `babel` is a set of `ini` files with basic definitions for about 200 languages. They are distributed as a separate `zip` file, not packed as `dtx`. With them, babel will fully support Unicode engines.

Most of them are essentially finished (except bugs and mistakes, of course). Some of them are still incomplete (but they will be usable), and there are some omissions (eg, Latin and polytonic Greek, and there are no geographic areas in Spanish). Hindi, French, Occitan and Breton will show a warning related to dates. Not all include LICR variants.

This is a preliminary documentation.

`ini` files contain the actual data; `tex` files are currently just proxies to the corresponding ini files. Most keys are self-explanatory.

charset the encoding used in the ini file.
version of the ini file
level "version" of the ini specification . which keys are available (they may grow in a compatible way) and how they should be read.
encodings a descriptive list of font encodings.
[captions] section of captions in the file charset
[captions.licr] same, but in pure ASCII using the LICR
date.long fields are as in the CLDR, but the syntax is different. Anything inside brackets is a date field (eg, MMMM for the month name) and anything outside is text. In addition, [ ] is a non breakable space and [.] is an abbreviation dot.

Keys may be further qualified in a particular language with a suffix starting with a uppercase letter. It can be just a letter (eg, `babel.name.A`, `babel.name.B`) or a name (eg, `date.long.Nominative`, `date.long.Formal`, but no language is currently using the latter). *Multi-letter* qualifiers are forward compatible in the sense they won't conflict with new "global" keys (which start always with a lowercase case). There is an exception, however: the section `counters` has been devised to have arbitrary keys, so you can add lowercased keys if you want.

7 Tools

1 ⟨⟨version=3.58.2366⟩⟩
2 ⟨⟨date=2021/05/08⟩⟩

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     \def\<##1>{\expandafter\noexpand\csname##1\endcsname}%
33     \edef\bbl@exp@aux{\endgroup#1}%
34   \bbl@exp@aux}
```

\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.

```
35 \def\bbl@tempa#1{%
36   \long\def\bbl@trim##1##2{%
37     \futurelet\bbl@trim@a\bbl@trim@c##2\@nil\@nil#1\@nil\relax{##1}}%
38   \def\bbl@trim@c{%
39     \ifx\bbl@trim@a\@sptoken
```

---

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

```
40        \expandafter\bbl@trim@b
41      \else
42        \expandafter\bbl@trim@b\expandafter#1%
43      \fi}%
44    \long\def\bbl@trim@b#1##1 \@nil{\bbl@trim@i##1}}
45  \bbl@tempa{ }
46  \long\def\bbl@trim@i#1\@nil#2\relax#3{#3{#1}}
47  \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 do not waste
memory.

```
48  \begingroup
49    \gdef\bbl@ifunset#1{%
50      \expandafter\ifx\csname#1\endcsname\relax
51        \expandafter\@firstoftwo
52      \else
53        \expandafter\@secondoftwo
54      \fi}
55    \bbl@ifunset{ifcsname}%
56      {}%
57      {\gdef\bbl@ifunset#1{%
58        \ifcsname#1\endcsname
59          \expandafter\ifx\csname#1\endcsname\relax
60            \bbl@afterelse\expandafter\@firstoftwo
61          \else
62            \bbl@afterfi\expandafter\@secondoftwo
63          \fi
64        \else
65          \expandafter\@firstoftwo
66        \fi}}
67  \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,

```
68  \def\bbl@ifblank#1{%
69    \bbl@ifblank@i#1\@nil\@nil\@secondoftwo\@firstoftwo\@nil}
70  \long\def\bbl@ifblank@i#1#2\@nil#3#4#5\@nil{#4}
71  \def\bbl@ifset#1#2#3{%
72    \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).

```
73  \def\bbl@forkv#1#2{%
74    \def\bbl@kvcmd##1##2##3{#2}%
75    \bbl@kvnext#1,\@nil,}
76  \def\bbl@kvnext#1,{%
77    \ifx\@nil#1\relax\else
78      \bbl@ifblank{#1}{}{\bbl@forkv@eq#1=\@empty=\@nil{#1}}%
79      \expandafter\bbl@kvnext
80    \fi}
81  \def\bbl@forkv@eq#1=#2=#3\@nil#4{%
82    \bbl@trim@def\bbl@forkv@a{#1}%
83    \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).

```
84 \def\bbl@vforeach#1#2{%
85   \def\bbl@forcmd##1{#2}%
86   \bbl@fornext#1,\@nil,}
87 \def\bbl@fornext#1,{%
88   \ifx\@nil#1\relax\else
89     \bbl@ifblank{#1}{}{\bbl@trim\bbl@forcmd{#1}}%
90     \expandafter\bbl@fornext
91   \fi}
92 \def\bbl@foreach#1{\expandafter\bbl@vforeach\expandafter{#1}}
```

\bbl@replace

```
93 \def\bbl@replace#1#2#3{% in #1 -> repl #2 by #3
94   \toks@{}%
95   \def\bbl@replace@aux##1#2##2#2{%
96     \ifx\bbl@nil##2%
97       \toks@\expandafter{\the\toks@##1}%
98     \else
99       \toks@\expandafter{\the\toks@##1#3}%
100      \bbl@afterfi
101      \bbl@replace@aux##2#2%
102    \fi}%
103  \expandafter\bbl@replace@aux#1#2\bbl@nil#2%
104  \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).

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

```
132 \def\bbl@ifsamestring#1#2{%
133   \begingroup
134     \protected@edef\bbl@tempb{#1}%
135     \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
136     \protected@edef\bbl@tempc{#2}%
137     \edef\bbl@tempc{\expandafter\strip@prefix\meaning\bbl@tempc}%
138     \ifx\bbl@tempb\bbl@tempc
139       \aftergroup\@firstoftwo
140     \else
141       \aftergroup\@secondoftwo
142     \fi
143   \endgroup}
144 \chardef\bbl@engine=%
145   \ifx\directlua\@undefined
146     \ifx\XeTeXinputencoding\@undefined
147       \z@
148     \else
149       \tw@
150     \fi
151   \else
152     \@ne
153   \fi
```

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

```
154 \def\bbl@bsphack{%
155   \ifhmode
156     \hskip\z@skip
157     \def\bbl@esphack{\loop\ifdim\lastskip>\z@\unskip\repeat\unskip}%
158   \else
159     \let\bbl@esphack\@empty
160   \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.

```
161 \def\bbl@cased{%
162   \ifx\oe\OE
163     \expandafter\in@\expandafter
164       {\expandafter\OE\expandafter}\expandafter{\oe}%
165     \ifin@
166       \bbl@afterelse\expandafter\MakeUppercase
167     \else
168       \bbl@afterfi\expandafter\MakeLowercase
169     \fi
170   \else
171     \expandafter\@firstofone
172   \fi}
173 ⟨⟨/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.

```
174 ⟨⟨*Make sure ProvidesFile is defined⟩⟩ ≡
175 \ifx\ProvidesFile\@undefined
176   \def\ProvidesFile#1[#2 #3 #4]{%
177     \wlog{File: #1 #4 #3 <#2>}%
178     \let\ProvidesFile\@undefined}
179 \fi
180 ⟨⟨/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.

```
181 ⟨⟨∗Define core switching macros⟩⟩ ≡
182 \ifx\language\@undefined
183   \csname newcount\endcsname\language
184 \fi
185 ⟨⟨/Define core switching macros⟩⟩
```

\last@language — Another counter is used to store the last language defined. For pre-3.0 formats an extra counter has to be allocated.

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

```
186 ⟨⟨∗Define core switching macros⟩⟩ ≡
187 ⟨⟨∗Define core switching macros⟩⟩ ≡
188 \countdef\last@language=19  % TODO. why? remove?
189 \def\addlanguage{\csname newlanguage\endcsname}
190 ⟨⟨/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 or LaTeX2.09. 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`)

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. The first two options are for debugging.

```
191 ⟨∗package⟩
192 \NeedsTeXFormat{LaTeX2e}[2005/12/01]
193 \ProvidesPackage{babel}[⟨⟨date⟩⟩ ⟨⟨version⟩⟩ The Babel package]
194 \@ifpackagewith{babel}{debug}
195   {\providecommand\bbl@trace[1]{\message{^^J[ #1 ]}}%
196    \let\bbl@debug\@firstofone
197    \ifx\directlua\@undefined\else
198      \directlua{ Babel = Babel or {}
199        Babel.debug = true }%
200    \fi}
201   {\providecommand\bbl@trace[1]{}%
202    \let\bbl@debug\@gobble
203    \ifx\directlua\@undefined\else
204      \directlua{ Babel = Babel or {}
205        Babel.debug = false }%
206    \fi}
207 ⟨⟨Basic macros⟩⟩
208   % Temporarily repeat here the code for errors. TODO.
209   \def\bbl@error#1#2{%
210     \begingroup
```

```
211        \def\\{\MessageBreak}%
212        \PackageError{babel}{#1}{#2}%
213      \endgroup}
214    \def\bbl@warning#1{%
215      \begingroup
216        \def\\{\MessageBreak}%
217        \PackageWarning{babel}{#1}%
218      \endgroup}
219    \def\bbl@infowarn#1{%
220      \begingroup
221        \def\\{\MessageBreak}%
222        \GenericWarning
223          {(babel) \@spaces\@spaces\@spaces}%
224          {Package babel Info: #1}%
225      \endgroup}
226    \def\bbl@info#1{%
227      \begingroup
228        \def\\{\MessageBreak}%
229        \PackageInfo{babel}{#1}%
230      \endgroup}
231  \def\bbl@nocaption{\protect\bbl@nocaption@i}
232  % TODO - Wrong for \today !!! Must be a separate macro.
233  \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
234    \global\@namedef{#2}{\textbf{?#1?}}%
235    \@nameuse{#2}%
236    \edef\bbl@tempa{#1}%
237    \bbl@sreplace\bbl@tempa{name}{}%
238    \bbl@warning{%
239      \@backslashchar#1 not set for '\languagename'. Please,\\%
240      define it after the language has been loaded\\%
241      (typically in the preamble) with\\%
242      \string\setlocalecaption{\languagename}{\bbl@tempa}{..}\\%
243      Reported}}
244  \def\bbl@tentative{\protect\bbl@tentative@i}
245  \def\bbl@tentative@i#1{%
246    \bbl@warning{%
247      Some functions for '#1' are tentative.\\%
248      They might not work as expected and their behavior\\%
249      may change in the future.\\%
250      Reported}}
251  \def\@nolanerr#1{%
252    \bbl@error
253      {You haven't defined the language #1\space yet.\\%
254       Perhaps you misspelled it or your installation\\%
255       is not complete}%
256      {Your command will be ignored, type <return> to proceed}}
257  \def\@nopatterns#1{%
258    \bbl@warning
259      {No hyphenation patterns were preloaded for\\%
260       the language `#1' into the format.\\%
261       Please, configure your TeX system to add them and\\%
262       rebuild the format. Now I will use the patterns\\%
263       preloaded for \bbl@nulllanguage\space instead}}
264      % End of errors
265  \@ifpackagewith{babel}{silent}
266    {\let\bbl@info\@gobble
267     \let\bbl@infowarn\@gobble
268     \let\bbl@warning\@gobble}
269    {}
```

```
270 %
271 \def\AfterBabelLanguage#1{%
272    \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.

```
273 \ifx\bbl@languages\@undefined\else
274    \begingroup
275      \catcode`\^^I=12
276      \@ifpackagewith{babel}{showlanguages}{%
277        \begingroup
278          \def\bbl@elt#1#2#3#4{\wlog{#2^^I#1^^I#3^^I#4}}%
279          \wlog{<*languages>}%
280          \bbl@languages
281          \wlog{</languages>}%
282        \endgroup}{}
283    \endgroup
284    \def\bbl@elt#1#2#3#4{%
285      \ifnum#2=\z@
286        \gdef\bbl@nulllanguage{#1}%
287        \def\bbl@elt##1##2##3##4{}%
288      \fi}%
289    \bbl@languages
290 \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.

```
291 \bbl@trace{Defining option 'base'}
292 \@ifpackagewith{babel}{base}{%
293    \let\bbl@onlyswitch\@empty
294    \let\bbl@provide@locale\relax
295    \input babel.def
296    \let\bbl@onlyswitch\@undefined
297    \ifx\directlua\@undefined
298      \DeclareOption*{\bbl@patterns{\CurrentOption}}%
299    \else
300      \input luababel.def
301      \DeclareOption*{\bbl@patterns@lua{\CurrentOption}}%
302    \fi
303    \DeclareOption{base}{}%
304    \DeclareOption{showlanguages}{}%
305    \ProcessOptions
306    \global\expandafter\let\csname opt@babel.sty\endcsname\relax
307    \global\expandafter\let\csname ver@babel.sty\endcsname\relax
308    \global\let\@ifl@ter@@\@ifl@ter
309    \def\@ifl@ter#1#2#3#4#5{\global\let\@ifl@ter\@ifl@ter@@}%
310    \endinput}{}%
311 % \end{macrocode}
312 %
313 % \subsection{\texttt{key=value} options and other general option}
314 %
315 %    The following macros extract language modifiers, and only real
316 %    package options are kept in the option list. Modifiers are saved
```

```
317 %    and assigned to |\BabelModifiers| at |\bbl@load@language|; when
318 %    no modifiers have been given, the former is |\relax|. How
319 %    modifiers are handled are left to language styles; they can use
320 %    |\in@|, loop them with |\@for| or load |keyval|, for example.
321 %
322 %    \begin{macrocode}
323 \bbl@trace{key=value and another general options}
324 \bbl@csarg\let{tempa\expandafter}\csname opt@babel.sty\endcsname
325 \def\bbl@tempb#1.#2{%  Remove trailing dot
326   #1\ifx\@empty#2\else,\bbl@afterfi\bbl@tempb#2\fi}%
327 \def\bbl@tempd#1.#2\@nnil{%  TODO. Refactor lists?
328   \ifx\@empty#2%
329     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
330   \else
331     \in@{,provide,}{,#1,}%
332     \ifin@
333       \edef\bbl@tempc{%
334         \ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.\bbl@tempb#2}%
335     \else
336       \in@{=}{#1}%
337       \ifin@
338         \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.#2}%
339       \else
340         \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
341         \bbl@csarg\edef{mod@#1}{\bbl@tempb#2}%
342       \fi
343     \fi
344   \fi}
345 \let\bbl@tempc\@empty
346 \bbl@foreach\bbl@tempa{\bbl@tempd#1.\@empty\@nnil}
347 \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.

```
348 \DeclareOption{KeepShorthandsActive}{}
349 \DeclareOption{activeacute}{}
350 \DeclareOption{activegrave}{}
351 \DeclareOption{debug}{}
352 \DeclareOption{noconfigs}{}
353 \DeclareOption{showlanguages}{}
354 \DeclareOption{silent}{}
355 \DeclareOption{mono}{}
356 \DeclareOption{shorthands=off}{\bbl@tempa shorthands=\bbl@tempa}
357 \chardef\bbl@iniflag\z@
358 \DeclareOption{provide=*}{\chardef\bbl@iniflag\@ne}     % main -> +1
359 \DeclareOption{provide+=*}{\chardef\bbl@iniflag\tw@}   % add = 2
360 \DeclareOption{provide*=*}{\chardef\bbl@iniflag\thr@@} % add + main
361 % A separate option
362 \let\bbl@autoload@options\@empty
363 \DeclareOption{provide@=*}{\def\bbl@autoload@options{import}}
364 % Don't use. Experimental. TODO.
365 \newif\ifbbl@single
366 \DeclareOption{selectors=off}{\bbl@singletrue}
367 ⟨⟨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.

```
368 \let\bbl@opt@shorthands\@nnil
369 \let\bbl@opt@config\@nnil
370 \let\bbl@opt@main\@nnil
371 \let\bbl@opt@headfoot\@nnil
372 \let\bbl@opt@layout\@nnil
```

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

```
373 \def\bbl@tempa#1=#2\bbl@tempa{%
374   \bbl@csarg\ifx{opt@#1}\@nnil
375     \bbl@csarg\edef{opt@#1}{#2}%
376   \else
377     \bbl@error
378     {Bad option `#1=#2'. Either you have misspelled the\\%
379      key or there is a previous setting of `#1'. Valid\\%
380      keys are, among others, `shorthands', `main', `bidi',\\%
381      `strings', `config', `headfoot', `safe', `math'.}%
382     {See the manual for further details.}
383   \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.

```
384 \let\bbl@language@opts\@empty
385 \DeclareOption*{%
386   \bbl@xin@{\string=}{\CurrentOption}%
387   \ifin@
388     \expandafter\bbl@tempa\CurrentOption\bbl@tempa
389   \else
390     \bbl@add@list\bbl@language@opts{\CurrentOption}%
391   \fi}
```

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

```
392 \ProcessOptions*
```

7.4 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=....

```
393 \bbl@trace{Conditional loading of shorthands}
394 \def\bbl@sh@string#1{%
395   \ifx#1\@empty\else
396     \ifx#1t\string~%
397     \else\ifx#1c\string,%
398     \else\string#1%
399     \fi\fi
400     \expandafter\bbl@sh@string
401   \fi}
402 \ifx\bbl@opt@shorthands\@nnil
403   \def\bbl@ifshorthand#1#2#3{#2}%
404 \else\ifx\bbl@opt@shorthands\@empty
405   \def\bbl@ifshorthand#1#2#3{#3}%
406 \else
```

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

```
407   \def\bbl@ifshorthand#1{%
```

```
408     \bbl@xin@{\string#1}{\bbl@opt@shorthands}%
409     \ifin@
410       \expandafter\@firstoftwo
411     \else
412       \expandafter\@secondoftwo
413     \fi}
```

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

```
414     \edef\bbl@opt@shorthands{%
415       \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.

```
416     \bbl@ifshorthand{'}%
417       {\PassOptionsToPackage{activeacute}{babel}}{}
418     \bbl@ifshorthand{`}%
419       {\PassOptionsToPackage{activegrave}{babel}}{}
420 \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.

```
421 \ifx\bbl@opt@headfoot\@nnil\else
422   \g@addto@macro\@resetactivechars{%
423     \set@typeset@protect
424     \expandafter\select@language@x\expandafter{\bbl@opt@headfoot}%
425     \let\protect\noexpand}
426 \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.

```
427 \ifx\bbl@opt@safe\@undefined
428   \def\bbl@opt@safe{BR}
429 \fi
430 \ifx\bbl@opt@main\@nnil\else
431   \edef\bbl@language@opts{%
432     \ifx\bbl@language@opts\@empty\else\bbl@language@opts,\fi
433       \bbl@opt@main}
434 \fi
```

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

```
435 \bbl@trace{Defining IfBabelLayout}
436 \ifx\bbl@opt@layout\@nnil
437   \newcommand\IfBabelLayout[3]{#3}%
438 \else
439   \newcommand\IfBabelLayout[1]{%
440     \@expandtwoargs\in@{.#1.}{.\bbl@opt@layout.}%
441     \ifin@
442       \expandafter\@firstoftwo
443     \else
444       \expandafter\@secondoftwo
445     \fi}
446 \fi
```

Common definitions. *In progress.* Still based on babel.def, but the code should be moved here.

```
447 \input babel.def
```

7.5 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.

```
448 ⟨⟨*More package options⟩⟩ ≡
449 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
450 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
451 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
452 ⟨⟨/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.

```
453 \bbl@trace{Cross referencing macros}
454 \ifx\bbl@opt@safe\@empty\else
455   \def\@newl@bel#1#2#3{%
456     {\@safe@activestrue
457     \bbl@ifunset{#1@#2}%
458       \relax
459       {\gdef\@multiplelabels{%
460         \@latex@warning@no@line{There were multiply-defined labels}}%
461       \@latex@warning@no@line{Label `#2' multiply defined}}%
462   \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.

```
463   \CheckCommand*\@testdef[3]{%
464     \def\reserved@a{#3}%
465     \expandafter\ifx\csname#1@#2\endcsname\reserved@a
466     \else
467       \@tempswatrue
468     \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.

```
469   \def\@testdef#1#2#3{%  TODO. With @samestring?
470     \@safe@activestrue
471     \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
472     \def\bbl@tempb{#3}%
473     \@safe@activesfalse
474     \ifx\bbl@tempa\relax
475     \else
476       \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
477     \fi
478     \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
479     \ifx\bbl@tempa\bbl@tempb
480     \else
481       \@tempswatrue
```

\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.

```
484 \bbl@xin@{R}\bbl@opt@safe
485 \ifin@
486    \bbl@redefinerobust\ref#1{%
487       \@safe@activestrue\org@ref{#1}\@safe@activesfalse}
488    \bbl@redefinerobust\pageref#1{%
489       \@safe@activestrue\org@pageref{#1}\@safe@activesfalse}
490 \else
491    \let\org@ref\ref
492    \let\org@pageref\pageref
493 \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.

```
494 \bbl@xin@{B}\bbl@opt@safe
495 \ifin@
496    \bbl@redefine\@citex[#1]#2{%
497       \@safe@activestrue\edef\@tempa{#2}\@safe@activesfalse
498       \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.

```
499    \AtBeginDocument{%
500       \@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.)

```
501       \def\@citex[#1][#2]#3{%
502          \@safe@activestrue\edef\@tempa{#3}\@safe@activesfalse
503          \org@@citex[#1][#2]{\@tempa}}%
504       }{}}
```

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

```
505    \AtBeginDocument{%
506       \@ifpackageloaded{cite}{%
507          \def\@citex[#1]#2{%
508             \@safe@activestrue\org@@citex[#1]{#2}\@safe@activesfalse}%
509          }{}}
```

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

```
510 \bbl@redefine\nocite#1{%
511    \@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.

```
512    \bbl@redefine\bibcite{%
513      \bbl@cite@choice
514      \bibcite}
```

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

```
515    \def\bbl@bibcite#1#2{%
516      \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.

```
517    \def\bbl@cite@choice{%
518      \global\let\bibcite\bbl@bibcite
519      \@ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{}%
520      \@ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{}%
521      \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.

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

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

```
523    \bbl@redefine\@bibitem#1{%
524      \@safe@activestrue\org@@bibitem{#1}\@safe@activesfalse}
525 \else
526   \let\org@nocite\nocite
527   \let\org@@citex\@citex
528   \let\org@bibcite\bibcite
529   \let\org@@bibitem\@bibitem
530 \fi
```

7.6 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.

```
531 \bbl@trace{Marks}
532 \IfBabelLayout{sectioning}
533   {\ifx\bbl@opt@headfoot\@nnil
534     \g@addto@macro\@resetactivechars{%
535       \set@typeset@protect
536       \expandafter\select@language@x\expandafter{\bbl@main@language}%
537       \let\protect\noexpand
538       \ifcase\bbl@bidimode\else % Only with bidi. See also above
539         \edef\thepage{%
540           \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
541       \fi}%
542   \fi}
543   {\ifbbl@single\else
544     \bbl@ifunset{markright }\bbl@redefine\bbl@redefinerobust
545     \markright#1{%
```

```
546        \bbl@ifblank{#1}%
547          {\org@markright{}}%
548          {\toks@{#1}%
549           \bbl@exp{%
550             \\\org@markright{\\\protect\\\foreignlanguage{\languagename}%
551               {\\\protect\\\bbl@restore@actives\the\toks@}}}}}%
```

The definition of \markboth is equivalent to that of \markright, except that we need two token 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.)

```
552        \ifx\@mkboth\markboth
553          \def\bbl@tempc{\let\@mkboth\markboth}
554        \else
555          \def\bbl@tempc{}
556        \fi
557      \bbl@ifunset{markboth }\bbl@redefine\bbl@redefinerobust
558      \markboth#1#2{%
559        \protected@edef\bbl@tempb##1{%
560          \protect\foreignlanguage
561          {\languagename}{\protect\bbl@restore@actives##1}}%
562        \bbl@ifblank{#1}%
563          {\toks@{}}%
564          {\toks@\expandafter{\bbl@tempb{#1}}}%
565        \bbl@ifblank{#2}%
566          {\@temptokena{}}%
567          {\@temptokena\expandafter{\bbl@tempb{#2}}}%
568        \bbl@exp{\\\org@markboth{\the\toks@}{\the\@temptokena}}}
569        \bbl@tempc
570      \fi}  % end ifbbl@single, end \IfBabelLayout
```

7.7 Preventing clashes with other packages

7.7.1 ifthen

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.

```
571 \bbl@trace{Preventing clashes with other packages}
572 \bbl@xin@{R}\bbl@opt@safe
573 \ifin@
574   \AtBeginDocument{%
575     \@ifpackageloaded{ifthen}{%
576       \bbl@redefine@long\ifthenelse#1#2#3{%
```

```
577          \let\bbl@temp@pref\pageref
578          \let\pageref\org@pageref
579          \let\bbl@temp@ref\ref
580          \let\ref\org@ref
581          \@safe@activestrue
582          \org@ifthenelse{#1}%
583            {\let\pageref\bbl@temp@pref
584             \let\ref\bbl@temp@ref
585             \@safe@activesfalse
586             #2}%
587            {\let\pageref\bbl@temp@pref
588             \let\ref\bbl@temp@ref
589             \@safe@activesfalse
590             #3}%
591          }%
592        }{}%
593      }
```

7.7.2 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.

```
594    \AtBeginDocument{%
595      \@ifpackageloaded{varioref}{%
596        \bbl@redefine\@@vpageref#1[#2]#3{%
597          \@safe@activestrue
598          \org@@@vpageref{#1}[#2]{#3}%
599          \@safe@activesfalse}%
600        \bbl@redefine\vrefpagenum#1#2{%
601          \@safe@activestrue
602          \org@vrefpagenum{#1}{#2}%
603          \@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.

```
604        \expandafter\def\csname Ref \endcsname#1{%
605          \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
606      }{}%
607    }
608 \fi
```

7.7.3 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.

```
609 \AtEndOfPackage{%
610    \AtBeginDocument{%
611      \@ifpackageloaded{hhline}%
612        {\expandafter\ifx\csname normal@char\string:\endcsname\relax
613         \else
614           \makeatletter
```

```
615          \def\@currname{hhline}\input{hhline.sty}\makeatother
616       \fi}%
617     {}}})
```

7.7.4 hyperref

\pdfstringdefDisableCommands A number of interworking problems between babel and hyperref are tackled by hyperref itself. The following code was introduced to prevent some annoying warnings but it broke bookmarks. This was quickly fixed in hyperref, which essentially made it no-op. However, it will not removed for the moment because hyperref is expecting it. TODO. Still true? Commented out in 2020/07/27.

```
618 % \AtBeginDocument{%
619 %   \ifx\pdfstringdefDisableCommands\@undefined\else
620 %     \pdfstringdefDisableCommands{\languageshorthands{system}}%
621 %   \fi}
```

7.7.5 fancyhdr

\FOREIGNLANGUAGE The package fancyhdr treats the running head and fout lines somewhat differently as the standard classes. A symptom of this is that the command \foreignlanguage which babel adds to the marks can end up inside the argument of \MakeUppercase. To prevent unexpected results we need to define \FOREIGNLANGUAGE here.

```
622 \DeclareRobustCommand{\FOREIGNLANGUAGE}[1]{%
623   \lowercase{\foreignlanguage{#1}}}
```

\substitutefontfamily 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. This command is deprecated. Use the tools provides by LaTeX.

```
624 \def\substitutefontfamily#1#2#3{%
625   \lowercase{\immediate\openout15=#1#2.fd\relax}%
626   \immediate\write15{%
627     \string\ProvidesFile{#1#2.fd}%
628     [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
629      \space generated font description file]^^J
630    \string\DeclareFontFamily{#1}{#2}{}^^J
631    \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{}^^J
632    \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{}^^J
633    \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{}^^J
634    \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{}^^J
635    \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{}^^J
636    \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{}^^J
637    \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{}^^J
638    \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{}^^J
639    }%
640   \closeout15
641   }
642 \@onlypreamble\substitutefontfamily
```

7.8 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. Unfortunately, fontenc deletes its package options, so we must guess which encodings has been loaded by traversing \@filelist to search for ⟨*enc*⟩enc.def. 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

```
643 \bbl@trace{Encoding and fonts}
```

78

```
644 \newcommand\BabelNonASCII{LGR,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU,PU,PD1}
645 \newcommand\BabelNonText{TS1,T3,TS3}
646 \let\org@TeX\TeX
647 \let\org@LaTeX\LaTeX
648 \let\ensureascii\@firstofone
649 \AtBeginDocument{%
650   \in@false
651   \bbl@foreach\BabelNonASCII{% is there a text non-ascii enc?
652     \ifin@\else
653       \lowercase{\bbl@xin@{,#1enc.def,}{,\@filelist,}}%
654     \fi}%
655   \ifin@ % if a text non-ascii has been loaded
656     \def\ensureascii#1{{\fontencoding{OT1}\selectfont#1}}%
657     \DeclareTextCommandDefault{\TeX}{\org@TeX}%
658     \DeclareTextCommandDefault{\LaTeX}{\org@LaTeX}%
659     \def\bbl@tempb#1\@@{\uppercase{\bbl@tempc#1}ENC.DEF\@empty\@@}%
660     \def\bbl@tempc#1ENC.DEF#2\@@{%
661       \ifx\@empty#2\else
662         \bbl@ifunset{T@#1}%
663           {}%
664           {\bbl@xin@{,#1,}{,\BabelNonASCII,\BabelNonText,}%
665            \ifin@
666              \DeclareTextCommand{\TeX}{#1}{\ensureascii{\org@TeX}}%
667              \DeclareTextCommand{\LaTeX}{#1}{\ensureascii{\org@LaTeX}}%
668            \else
669              \def\ensureascii##1{{\fontencoding{#1}\selectfont##1}}%
670            \fi}%
671       \fi}%
672     \bbl@foreach\@filelist{\bbl@tempb#1\@@}%  TODO - \@@ de mas??
673     \bbl@xin@{,\cf@encoding,}{,\BabelNonASCII,\BabelNonText,}%
674     \ifin@\else
675       \edef\ensureascii#1{{%
676         \noexpand\fontencoding{\cf@encoding}\noexpand\selectfont#1}}%
677     \fi
678   \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.

```
679 \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.

```
680 \AtBeginDocument{%
681   \@ifpackageloaded{fontspec}%
682     {\xdef\latinencoding{%
683        \ifx\UTFencname\@undefined
684          EU\ifcase\bbl@engine\or2\or1\fi
685        \else
686          \UTFencname
687        \fi}}%
688     {\gdef\latinencoding{OT1}%
689      \ifx\cf@encoding\bbl@t@one
690        \xdef\latinencoding{\bbl@t@one}%
```

```
691      \else
692        \ifx\@fontenc@load@list\@undefined
693          \@ifl@aded{def}{t1enc}{\xdef\latinencoding{\bbl@t@one}}{}%
694        \else
695          \def\@elt#1{,#1,}%
696          \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
697          \let\@elt\relax
698          \bbl@xin@{,T1,}\bbl@tempa
699          \ifin@
700            \xdef\latinencoding{\bbl@t@one}%
701          \fi
702        \fi
703      \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.

```
704 \DeclareRobustCommand{\latintext}{%
705   \fontencoding{\latinencoding}\selectfont
706   \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.

```
707 \ifx\@undefined\DeclareTextFontCommand
708   \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
709 \else
710   \DeclareTextFontCommand{\textlatin}{\latintext}
711 \fi
```

7.9 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.

As a frist 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.

```
712 \ifodd\bbl@engine
713   \def\bbl@activate@preotf{%
714     \let\bbl@activate@preotf\relax  % only once
715     \directlua{
```

```
716        Babel = Babel or {}
717        %
718        function Babel.pre_otfload_v(head)
719          if Babel.numbers and Babel.digits_mapped then
720            head = Babel.numbers(head)
721          end
722          if Babel.bidi_enabled then
723            head = Babel.bidi(head, false, dir)
724          end
725          return head
726        end
727        %
728        function Babel.pre_otfload_h(head, gc, sz, pt, dir)
729          if Babel.numbers and Babel.digits_mapped then
730            head = Babel.numbers(head)
731          end
732          if Babel.bidi_enabled then
733            head = Babel.bidi(head, false, dir)
734          end
735          return head
736        end
737        %
738        luatexbase.add_to_callback('pre_linebreak_filter',
739          Babel.pre_otfload_v,
740          'Babel.pre_otfload_v',
741          luatexbase.priority_in_callback('pre_linebreak_filter',
742            'luaotfload.node_processor') or nil)
743        %
744        luatexbase.add_to_callback('hpack_filter',
745          Babel.pre_otfload_h,
746          'Babel.pre_otfload_h',
747          luatexbase.priority_in_callback('hpack_filter',
748            'luaotfload.node_processor') or nil)
749    }}
750 \fi
```

The basic setup. In luatex, the output is modified at a very low level to set the \bodydir to the
\pagedir.

```
751 \bbl@trace{Loading basic (internal) bidi support}
752 \ifodd\bbl@engine
753   \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
754     \let\bbl@beforeforeign\leavevmode
755     \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
756     \RequirePackage{luatexbase}
757     \bbl@activate@preotf
758     \directlua{
759       require('babel-data-bidi.lua')
760       \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
761         require('babel-bidi-basic.lua')
762       \or
763         require('babel-bidi-basic-r.lua')
764       \fi}
765     % TODO - to locale_props, not as separate attribute
766     \newattribute\bbl@attr@dir
767     % TODO. I don't like it, hackish:
768     \bbl@exp{\output{\bodydir\pagedir\the\output}}
769     \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
770   \fi\fi
771 \else
```

```
772  \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
773    \bbl@error
774      {The bidi method `basic' is available only in\\%
775       luatex. I'll continue with `bidi=default', so\\%
776       expect wrong results}%
777      {See the manual for further details.}%
778    \let\bbl@beforeforeign\leavevmode
779    \AtEndOfPackage{%
780      \EnableBabelHook{babel-bidi}%
781      \bbl@xebidipar}
782  \fi\fi
783  \def\bbl@loadxebidi#1{%
784    \ifx\RTLfootnotetext\@undefined
785      \AtEndOfPackage{%
786        \EnableBabelHook{babel-bidi}%
787        \ifx\fontspec\@undefined
788          \bbl@loadfontspec % bidi needs fontspec
789        \fi
790        \usepackage#1{bidi}}%
791    \fi}
792  \ifnum\bbl@bidimode>200
793    \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
794      \bbl@tentative{bidi=bidi}
795      \bbl@loadxebidi{}
796    \or
797      \bbl@loadxebidi{[rldocument]}
798    \or
799      \bbl@loadxebidi{}
800    \fi
801  \fi
802 \fi
803 \ifnum\bbl@bidimode=\@ne
804   \let\bbl@beforeforeign\leavevmode
805   \ifodd\bbl@engine
806     \newattribute\bbl@attr@dir
807     \bbl@exp{\output{\bodydir\pagedir\the\output}}%
808   \fi
809   \AtEndOfPackage{%
810     \EnableBabelHook{babel-bidi}%
811     \ifodd\bbl@engine\else
812       \bbl@xebidipar
813     \fi}
814 \fi
```

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

```
815 \bbl@trace{Macros to switch the text direction}
816 \def\bbl@alscripts{,Arabic,Syriac,Thaana,}
817 \def\bbl@rscripts{% TODO. Base on codes ??
818   ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
819   Old Hungarian,Old Hungarian,Lydian,Mandaean,Manichaean,%
820   Manichaean,Meroitic Cursive,Meroitic,Old North Arabian,%
821   Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
822   Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
823   Old South Arabian,}%
824 \def\bbl@provide@dirs#1{%
825   \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
826   \ifin@
827     \global\bbl@csarg\chardef{wdir@#1}\@ne
```

```
828    \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
829    \ifin@
830      \global\bbl@csarg\chardef{wdir@#1}\tw@  % useless in xetex
831    \fi
832  \else
833    \global\bbl@csarg\chardef{wdir@#1}\z@
834  \fi
835  \ifodd\bbl@engine
836    \bbl@csarg\ifcase{wdir@#1}%
837      \directlua{ Babel.locale_props[\the\localeid].textdir = 'l' }%
838    \or
839      \directlua{ Babel.locale_props[\the\localeid].textdir = 'r' }%
840    \or
841      \directlua{ Babel.locale_props[\the\localeid].textdir = 'al' }%
842    \fi
843  \fi}
844 \def\bbl@switchdir{%
845  \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
846  \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
847  \bbl@exp{\\\bbl@setdirs\bbl@cl{wdir}}}
848 \def\bbl@setdirs#1{% TODO - math
849  \ifcase\bbl@select@type % TODO - strictly, not the right test
850    \bbl@bodydir{#1}%
851    \bbl@pardir{#1}%
852  \fi
853  \bbl@textdir{#1}}
854 % TODO. Only if \bbl@bidimode > 0?:
855 \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
856 \DisableBabelHook{babel-bidi}
```

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

```
857 \ifodd\bbl@engine  % luatex=1
858    \chardef\bbl@thetextdir\z@
859    \chardef\bbl@thepardir\z@
860    \def\bbl@getluadir#1{%
861      \directlua{
862        if tex.#1dir == 'TLT' then
863          tex.sprint('0')
864        elseif tex.#1dir == 'TRT' then
865          tex.sprint('1')
866        end}}
867    \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
868      \ifcase#3\relax
869        \ifcase\bbl@getluadir{#1}\relax\else
870          #2 TLT\relax
871        \fi
872      \else
873        \ifcase\bbl@getluadir{#1}\relax
874          #2 TRT\relax
875        \fi
876      \fi}
877    \def\bbl@textdir#1{%
878      \bbl@setluadir{text}\textdir{#1}%
879      \chardef\bbl@thetextdir#1\relax
880      \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*3+#1}}
881    \def\bbl@pardir#1{%
882      \bbl@setluadir{par}\pardir{#1}%
883      \chardef\bbl@thepardir#1\relax}
884    \def\bbl@bodydir{\bbl@setluadir{body}\bodydir}
```

```
885  \def\bbl@pagedir{\bbl@setluadir{page}\pagedir}
886  \def\bbl@dirparastext{\pardir\the\textdir\relax}%    %%%%
887  % Sadly, we have to deal with boxes in math with basic.
888  % Activated every math with the package option bidi=:
889  \def\bbl@mathboxdir{%
890    \ifcase\bbl@thetextdir\relax
891      \everyhbox{\textdir TLT\relax}%
892    \else
893      \everyhbox{\textdir TRT\relax}%
894    \fi}
895  \frozen@everymath\expandafter{%
896    \expandafter\bbl@mathboxdir\the\frozen@everymath}
897  \frozen@everydisplay\expandafter{%
898    \expandafter\bbl@mathboxdir\the\frozen@everydisplay}
899 \else % pdftex=0, xetex=2
900    \newcount\bbl@dirlevel
901    \chardef\bbl@thetextdir\z@
902    \chardef\bbl@thepardir\z@
903    \def\bbl@textdir#1{%
904      \ifcase#1\relax
905        \chardef\bbl@thetextdir\z@
906        \bbl@textdir@i\beginL\endL
907       \else
908        \chardef\bbl@thetextdir\@ne
909        \bbl@textdir@i\beginR\endR
910      \fi}
911    \def\bbl@textdir@i#1#2{%
912      \ifhmode
913        \ifnum\currentgrouplevel>\z@
914          \ifnum\currentgrouplevel=\bbl@dirlevel
915            \bbl@error{Multiple bidi settings inside a group}%
916              {I'll insert a new group, but expect wrong results.}%
917            \bgroup\aftergroup#2\aftergroup\egroup
918          \else
919            \ifcase\currentgrouptype\or % 0 bottom
920              \aftergroup#2% 1 simple {}
921            \or
922              \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
923            \or
924              \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
925            \or\or\or % vbox vtop align
926            \or
927              \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
928            \or\or\or\or\or\or % output math disc insert vcent mathchoice
929            \or
930              \aftergroup#2% 14 \begingroup
931            \else
932              \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
933            \fi
934          \fi
935          \bbl@dirlevel\currentgrouplevel
936        \fi
937        #1%
938      \fi}
939    \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
940    \let\bbl@bodydir\@gobble
941    \let\bbl@pagedir\@gobble
942    \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).

```
943  \def\bbl@xebidipar{%
944    \let\bbl@xebidipar\relax
945    \TeXXeTstate\@ne
946    \def\bbl@xeeverypar{%
947      \ifcase\bbl@thepardir
948        \ifcase\bbl@thetextdir\else\beginR\fi
949      \else
950        {\setbox\z@\lastbox\beginR\box\z@}%
951      \fi}%
952    \let\bbl@severypar\everypar
953    \newtoks\everypar
954    \everypar=\bbl@severypar
955    \bbl@severypar{\bbl@xeeverypar\the\everypar}}
956  \ifnum\bbl@bidimode>200
957    \let\bbl@textdir@i\@gobbletwo
958    \let\bbl@xebidipar\@empty
959    \AddBabelHook{bidi}{foreign}{%
960      \def\bbl@tempa{\def\BabelText####1}%
961      \ifcase\bbl@thetextdir
962        \expandafter\bbl@tempa\expandafter{\BabelText{\LR{##1}}}%
963      \else
964        \expandafter\bbl@tempa\expandafter{\BabelText{\RL{##1}}}%
965      \fi}
966    \def\bbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}
967  \fi
968 \fi
```

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

```
969 \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}
970 \AtBeginDocument{%
971   \ifx\pdfstringdefDisableCommands\@undefined\else
972     \ifx\pdfstringdefDisableCommands\relax\else
973       \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
974     \fi
975   \fi}
```

7.10 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.

```
976 \bbl@trace{Local Language Configuration}
977 \ifx\loadlocalcfg\@undefined
978   \@ifpackagewith{babel}{noconfigs}%
979     {\let\loadlocalcfg\@gobble}%
980     {\def\loadlocalcfg#1{%
981       \InputIfFileExists{#1.cfg}%
982         {\typeout{*************************************^^J%
983                   * Local config file #1.cfg used^^J%
984                   *}}%
985       \@empty}}
986 \fi
```

Just to be compatible with LaTeX 2.09 we add a few more lines of code. TODO. Necessary? Correct place? Used by some ldf file?

```
987 \ifx\@unexpandable@protect\@undefined
988   \def\@unexpandable@protect{\noexpand\protect\noexpand}
989   \long\def\protected@write#1#2#3{%
990     \begingroup
991       \let\thepage\relax
992       #2%
993       \let\protect\@unexpandable@protect
994       \edef\reserved@a{\write#1{#3}}%
995       \reserved@a
996     \endgroup
997     \if@nobreak\ifvmode\nobreak\fi\fi}
998 \fi
999 %
1000 % \subsection{Language options}
1001 %
1002 % Languages are loaded when processing the corresponding option
1003 % \textit{except} if a |main| language has been set. In such a
1004 % case, it is not loaded until all options has been processed.
1005 % The following macro inputs the ldf file and does some additional
1006 % checks (|\input| works, too, but possible errors are not catched).
1007 %
1008 %     \begin{macrocode}
1009 \bbl@trace{Language options}
1010 \let\bbl@afterlang\relax
1011 \let\BabelModifiers\relax
1012 \let\bbl@loaded\@empty
1013 \def\bbl@load@language#1{%
1014   \InputIfFileExists{#1.ldf}%
1015     {\edef\bbl@loaded{\CurrentOption
1016        \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}%
1017     \expandafter\let\expandafter\bbl@afterlang
1018        \csname\CurrentOption.ldf-h@@k\endcsname
1019     \expandafter\let\expandafter\BabelModifiers
1020        \csname bbl@mod@\CurrentOption\endcsname}%
1021     {\bbl@error{%
1022       Unknown option `\CurrentOption'. Either you misspelled it\\%
1023       or the language definition file \CurrentOption.ldf was not found}{%
1024       Valid options are, among others: shorthands=, KeepShorthandsActive,\\%
1025       activeacute, activegrave, noconfigs, safe=, main=, math=\\%
1026       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.

```
1027 \def\bbl@try@load@lang#1#2#3{%
1028   \IfFileExists{\CurrentOption.ldf}%
1029     {\bbl@load@language{\CurrentOption}}%
1030     {#1\bbl@load@language{#2}#3}}
1031 \DeclareOption{hebrew}{%
1032   \input{rlbabel.def}%
1033   \bbl@load@language{hebrew}}
1034 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}{}}
1035 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}{}}
1036 \DeclareOption{nynorsk}{\bbl@try@load@lang{}{norsk}{}}
1037 \DeclareOption{polutonikogreek}{%
1038   \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}
1039 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}{}}
```

```
1040 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}{}}
1041 \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.

```
1042 \ifx\bbl@opt@config\@nnil
1043   \@ifpackagewith{babel}{noconfigs}{}%
1044     {\InputIfFileExists{bblopts.cfg}%
1045       {\typeout{*************************************^^J%
1046               * Local config file bblopts.cfg used^^J%
1047               *}}%
1048       {}}%
1049 \else
1050   \InputIfFileExists{\bbl@opt@config.cfg}%
1051     {\typeout{*************************************^^J%
1052               * Local config file \bbl@opt@config.cfg used^^J%
1053               *}}%
1054     {\bbl@error{%
1055        Local config file `\bbl@opt@config.cfg' not found}{%
1056        Perhaps you misspelled it.}}%
1057 \fi
```

Recognizing global options in packages not having a closed set of them is not trivial, as for them to be
processed they must be defined explicitly. So, package options not yet taken into account and stored
in bbl@language@opts are assumed to be languages (note this list also contains the language given
with main). If not declared above, the names of the option and the file are the same.

```
1058 \let\bbl@tempc\relax
1059 \bbl@foreach\bbl@language@opts{%
1060   \ifcase\bbl@iniflag  % Default
1061     \bbl@ifunset{ds@#1}%
1062       {\DeclareOption{#1}{\bbl@load@language{#1}}}%
1063       {}%
1064   \or     % provide=*
1065     \@gobble % case 2 same as 1
1066   \or     % provide+=*
1067     \bbl@ifunset{ds@#1}%
1068       {\IfFileExists{#1.ldf}{}%
1069         {\IfFileExists{babel-#1.tex}{}{\@namedef{ds@#1}{}}}}%
1070       {}%
1071     \bbl@ifunset{ds@#1}%
1072       {\def\bbl@tempc{#1}%
1073        \DeclareOption{#1}{%
1074          \ifnum\bbl@iniflag>\@ne
1075            \bbl@ldfinit
1076            \babelprovide[import]{#1}%
1077            \bbl@afterldf{}%
1078          \else
1079            \bbl@load@language{#1}%
1080          \fi}}%
1081       {}%
1082   \or     % provide*=*
1083     \def\bbl@tempc{#1}%
1084     \bbl@ifunset{ds@#1}%
1085       {\DeclareOption{#1}{%
1086          \bbl@ldfinit
1087          \babelprovide[import]{#1}%
1088          \bbl@afterldf{}}}%
```

```
1089        {}%
1090    \fi}
```

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

```
1091 \let\bbl@tempb\@nnil
1092 \bbl@foreach\@classoptionslist{%
1093    \bbl@ifunset{ds@#1}%
1094      {\IfFileExists{#1.ldf}{}%
1095        {\IfFileExists{babel-#1.tex}{}{\@namedef{ds@#1}{}}}}%
1096      {}%
1097    \bbl@ifunset{ds@#1}%
1098      {\def\bbl@tempb{#1}%
1099       \DeclareOption{#1}{%
1100         \ifnum\bbl@iniflag>\@ne
1101           \bbl@ldfinit
1102           \babelprovide[import]{#1}%
1103           \bbl@afterldf{}%
1104         \else
1105           \bbl@load@language{#1}%
1106         \fi}}%
1107      {}}
```

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

```
1108 \ifnum\bbl@iniflag=\z@\else
1109    \ifx\bbl@opt@main\@nnil
1110      \ifx\bbl@tempc\relax
1111        \let\bbl@opt@main\bbl@tempb
1112      \else
1113        \let\bbl@opt@main\bbl@tempc
1114      \fi
1115    \fi
1116 \fi
1117 \ifx\bbl@opt@main\@nnil\else
1118    \expandafter
1119    \let\expandafter\bbl@loadmain\csname ds@\bbl@opt@main\endcsname
1120    \expandafter\let\csname ds@\bbl@opt@main\endcsname\@empty
1121 \fi
```

And we are done, because all options for this pass has been declared. Those already processed in the first pass are just ignored.

The options have to be processed in the order in which the user specified them (except, of course, global options, which LaTeX processes before):

```
1122 \def\AfterBabelLanguage#1{%
1123    \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang}{}}
1124 \DeclareOption*{}
1125 \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.

```
1126 \bbl@trace{Option 'main'}
1127 \ifx\bbl@opt@main\@nnil
1128    \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}
1129    \let\bbl@tempc\@empty
1130    \bbl@for\bbl@tempb\bbl@tempa{%
1131      \bbl@xin@{,\bbl@tempb,}{,\bbl@loaded,}%
```

```
1132    \ifin@\edef\bbl@tempc{\bbl@tempb}\fi}
1133  \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
1134  \expandafter\bbl@tempa\bbl@loaded,\@nnil
1135  \ifx\bbl@tempb\bbl@tempc\else
1136    \bbl@warning{%
1137      Last declared language option is `\bbl@tempc',\\%
1138      but the last processed one was `\bbl@tempb'.\\%
1139      The main language cannot be set as both a global\\%
1140      and a package option. Use `main=\bbl@tempc' as\\%
1141      option. Reported}%
1142    \fi
1143 \else
1144  \ifodd\bbl@iniflag  % case 1,3
1145    \bbl@ldfinit
1146    \let\CurrentOption\bbl@opt@main
1147    \bbl@exp{\\\babelprovide[import,main]{\bbl@opt@main}}
1148    \bbl@afterldf{}%
1149  \else % case 0,2
1150    \chardef\bbl@iniflag\z@  % Force ldf
1151    \expandafter\let\csname ds@\bbl@opt@main\endcsname\bbl@loadmain
1152    \ExecuteOptions{\bbl@opt@main}
1153    \DeclareOption*{}%
1154    \ProcessOptions*
1155    \fi
1156 \fi
1157 \def\AfterBabelLanguage{%
1158  \bbl@error
1159    {Too late for \string\AfterBabelLanguage}%
1160    {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.

```
1161 \ifx\bbl@main@language\@undefined
1162  \bbl@info{%
1163    You haven't specified a language. I'll use 'nil'\\%
1164    as the main language. Reported}
1165    \bbl@load@language{nil}
1166 \fi
1167 ⟨/package⟩
1168 ⟨∗core⟩
```

8   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.

8.1   Tools

```
1169 \ifx\ldf@quit\@undefined\else
```

```
1170 \endinput\fi % Same line!
1171 ⟨⟨Make sure ProvidesFile is defined⟩⟩
1172 \ProvidesFile{babel.def}[⟨⟨date⟩⟩ ⟨⟨version⟩⟩ Babel common definitions]
```

The file babel.def expects some definitions made in the LaTeX 2ε style file. So, In LaTeX2.09 and 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).

```
1173 \ifx\AtBeginDocument\@undefined  % TODO. change test.
1174   ⟨⟨Emulate LaTeX⟩⟩
1175   \def\languagename{english}%
1176   \let\bbl@opt@shorthands\@nnil
1177   \def\bbl@ifshorthand#1#2#3{#2}%
1178   \let\bbl@language@opts\@empty
1179   \ifx\babeloptionstrings\@undefined
1180     \let\bbl@opt@strings\@nnil
1181   \else
1182     \let\bbl@opt@strings\babeloptionstrings
1183   \fi
1184   \def\BabelStringsDefault{generic}
1185   \def\bbl@tempa{normal}
1186   \ifx\babeloptionmath\bbl@tempa
1187     \def\bbl@mathnormal{\noexpand\textormath}
1188   \fi
1189   \def\AfterBabelLanguage#1#2{}
1190   \ifx\BabelModifiers\@undefined\let\BabelModifiers\relax\fi
1191   \let\bbl@afterlang\relax
1192   \def\bbl@opt@safe{BR}
1193   \ifx\@uclclist\@undefined\let\@uclclist\@empty\fi
1194   \ifx\bbl@trace\@undefined\def\bbl@trace#1{}\fi
1195   \expandafter\newif\csname ifbbl@single\endcsname
1196   \chardef\bbl@bidimode\z@
1197 \fi
```

Exit immediately with 2.09. An error is raised by the sty file, but also try to minimize the number of errors.

```
1198 \ifx\bbl@trace\@undefined
1199   \let\LdfInit\endinput
1200   \def\ProvidesLanguage#1{\endinput}
1201 \endinput\fi % Same line!
```

And continue.

9   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.

```
1202 ⟨⟨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.

```
1203 \def\bbl@version{⟨⟨version⟩⟩}
1204 \def\bbl@date{⟨⟨date⟩⟩}
1205 \def\adddialect#1#2{%
1206   \global\chardef#1#2\relax
1207   \bbl@usehooks{adddialect}{{#1}{#2}}%
1208   \begingroup
1209     \count@#1\relax
```

```
1210    \def\bbl@elt##1##2##3##4{%
1211      \ifnum\count@=##2\relax
1212        \edef\bbl@tempa{\expandafter\@gobbletwo\string#1}%
1213        \bbl@info{Hyphen rules for '\expandafter\@gobble\bbl@tempa'
1214                  set to \expandafter\string\csname l@##1\endcsname\\%
1215                  (\string\language\the\count@). Reported}%
1216        \def\bbl@elt####1####2####3####4{}%
1217      \fi}%
1218    \bbl@cs{languages}%
1219  \endgroup}
```

\bbl@iflanguage executes code only if the language l@ exists. Otherwise raises and 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 intented 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.

```
1220 \def\bbl@fixname#1{%
1221   \begingroup
1222     \def\bbl@tempe{l@}%
1223     \edef\bbl@tempd{\noexpand\@ifundefined{\noexpand\bbl@tempe#1}}%
1224     \bbl@tempd
1225       {\lowercase\expandafter{\bbl@tempd}%
1226         {\uppercase\expandafter{\bbl@tempd}%
1227           \@empty
1228           {\edef\bbl@tempd{\def\noexpand#1{#1}}%
1229            \uppercase\expandafter{\bbl@tempd}}}%
1230         {\edef\bbl@tempd{\def\noexpand#1{#1}}%
1231          \lowercase\expandafter{\bbl@tempd}}}%
1232       \@empty
1233     \edef\bbl@tempd{\endgroup\def\noexpand#1{#1}}%
1234   \bbl@tempd
1235   \bbl@exp{\\\bbl@usehooks{languagename}{{\languagename}{#1}}}}}
1236 \def\bbl@iflanguage#1{%
1237   \@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.

```
1238 \def\bbl@bcpcase#1#2#3#4\@@#5{%
1239   \ifx\@empty#3%
1240     \uppercase{\def#5{#1#2}}%
1241   \else
1242     \uppercase{\def#5{#1}}%
1243     \lowercase{\edef#5{#5#2#3#4}}%
1244   \fi}
1245 \def\bbl@bcplookup#1-#2-#3-#4\@@{%
1246   \let\bbl@bcp\relax
1247   \lowercase{\def\bbl@tempa{#1}}%
1248   \ifx\@empty#2%
1249     \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
1250   \else\ifx\@empty#3%
1251     \bbl@bcpcase#2\@empty\@empty\@@\bbl@tempb
1252     \IfFileExists{babel-\bbl@tempa-\bbl@tempb.ini}%
1253       {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb}}%
1254       {}%
1255     \ifx\bbl@bcp\relax
```

```
1256        \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
1257      \fi
1258    \else
1259      \bbl@bcpcase#2\@empty\@empty\@@\bbl@tempb
1260      \bbl@bcpcase#3\@empty\@empty\@@\bbl@tempc
1261      \IfFileExists{babel-\bbl@tempa-\bbl@tempb-\bbl@tempc.ini}%
1262        {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb-\bbl@tempc}}%
1263        {}%
1264      \ifx\bbl@bcp\relax
1265        \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
1266          {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
1267          {}%
1268      \fi
1269      \ifx\bbl@bcp\relax
1270        \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
1271          {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
1272          {}%
1273      \fi
1274      \ifx\bbl@bcp\relax
1275        \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
1276      \fi
1277    \fi\fi}
1278 \let\bbl@initoload\relax
1279 \def\bbl@provide@locale{%
1280    \ifx\babelprovide\@undefined
1281      \bbl@error{For a language to be defined on the fly 'base'\\%
1282                 is not enough, and the whole package must be\\%
1283                 loaded. Either delete the 'base' option or\\%
1284                 request the languages explicitly}%
1285                {See the manual for further details.}%
1286    \fi
1287 % TODO. Option to search if loaded, with \LocaleForEach
1288    \let\bbl@auxname\languagename % Still necessary. TODO
1289    \bbl@ifunset{bbl@bcp@map@\languagename}{}% Move uplevel??
1290      {\edef\languagename{\@nameuse{bbl@bcp@map@\languagename}}}%
1291    \ifbbl@bcpallowed
1292      \expandafter\ifx\csname date\languagename\endcsname\relax
1293        \expandafter
1294        \bbl@bcplookup\languagename-\@empty-\@empty-\@empty\@@
1295        \ifx\bbl@bcp\relax\else  % Returned by \bbl@bcplookup
1296          \edef\languagename{\bbl@bcp@prefix\bbl@bcp}%
1297          \edef\localename{\bbl@bcp@prefix\bbl@bcp}%
1298          \expandafter\ifx\csname date\languagename\endcsname\relax
1299            \let\bbl@initoload\bbl@bcp
1300            \bbl@exp{\\\babelprovide[\bbl@autoload@bcpoptions]{\languagename}}%
1301            \let\bbl@initoload\relax
1302          \fi
1303          \bbl@csarg\xdef{bcp@map@\bbl@bcp}{\localename}%
1304        \fi
1305      \fi
1306    \fi
1307    \expandafter\ifx\csname date\languagename\endcsname\relax
1308      \IfFileExists{babel-\languagename.tex}%
1309        {\bbl@exp{\\\babelprovide[\bbl@autoload@options]{\languagename}}}%
1310        {}%
1311    \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.

```
1312 \def\iflanguage#1{%
1313   \bbl@iflanguage{#1}{%
1314     \ifnum\csname l@#1\endcsname=\language
1315       \expandafter\@firstoftwo
1316     \else
1317       \expandafter\@secondoftwo
1318     \fi}}
```

9.1 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.

```
1319 \let\bbl@select@type\z@
1320 \edef\selectlanguage{%
1321   \noexpand\protect
1322   \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.

```
1323 \ifx\@undefined\protect\let\protect\relax\fi
```

The following definition is preserved for backwards compatibility. It is related to a trick for 2.09.

```
1324 \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.

```
1325 \def\bbl@language@stack{}
```

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

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

```
1326 \def\bbl@push@language{%
1327   \ifx\languagename\@undefined\else
1328     \xdef\bbl@language@stack{\languagename+\bbl@language@stack}%
1329   \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.

```
1330 \def\bbl@pop@lang#1+#2\@@{%
1331   \edef\languagename{#1}%
1332   \xdef\bbl@language@stack{#2}}
```

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

```
1333 \let\bbl@ifrestoring\@secondoftwo
1334 \def\bbl@pop@language{%
1335   \expandafter\bbl@pop@lang\bbl@language@stack\@@
1336   \let\bbl@ifrestoring\@firstoftwo
1337   \expandafter\bbl@set@language\expandafter{\languagename}%
1338   \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).

```
1339 \chardef\localeid\z@
1340 \def\bbl@id@last{0}    % No real need for a new counter
1341 \def\bbl@id@assign{%
1342   \bbl@ifunset{bbl@id@@\languagename}%
1343     {\count@\bbl@id@last\relax
1344      \advance\count@\@ne
1345      \bbl@csarg\chardef{id@@\languagename}\count@
1346      \edef\bbl@id@last{\the\count@}%
1347      \ifcase\bbl@engine\or
1348        \directlua{
1349          Babel = Babel or {}
1350          Babel.locale_props = Babel.locale_props or {}
1351          Babel.locale_props[\bbl@id@last] = {}
1352          Babel.locale_props[\bbl@id@last].name = '\languagename'
1353        }%
1354      \fi}%
1355     {}%
1356   \chardef\localeid\bbl@cl{id@}}
```

The unprotected part of \selectlanguage.

```
1357 \expandafter\def\csname selectlanguage \endcsname#1{%
1358   \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\tw@\fi
1359   \bbl@push@language
1360   \aftergroup\bbl@pop@language
1361   \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.

```
1362 \def\BabelContentsFiles{toc,lof,lot}
1363 \def\bbl@set@language#1{% from selectlanguage, pop@
1364   % The old buggy way. Preserved for compatibility.
1365   \edef\languagename{%
1366     \ifnum\escapechar=\expandafter`\string#1\@empty
1367     \else\string#1\@empty\fi}%
```

94

```
1368    \ifcat\relax\noexpand#1%
1369      \expandafter\ifx\csname date\languagename\endcsname\relax
1370        \edef\languagename{#1}%
1371        \let\localename\languagename
1372      \else
1373        \bbl@info{Using '\string\language' instead of 'language' is\\%
1374                  deprecated. If what you want is to use a\\%
1375                  macro containing the actual locale, make\\%
1376                  sure it does not not match any language.\\%
1377                  Reported}%
1378 %               I'll\\%
1379 %               try to fix '\string\localename', but I cannot promise\\%
1380 %               anything. Reported}%
1381        \ifx\scantokens\@undefined
1382          \def\localename{??}%
1383        \else
1384          \scantokens\expandafter{\expandafter
1385            \def\expandafter\localename\expandafter{\languagename}}%
1386        \fi
1387      \fi
1388    \else
1389      \def\localename{#1}% This one has the correct catcodes
1390    \fi
1391    \select@language{\languagename}%
1392    % write to auxs
1393    \expandafter\ifx\csname date\languagename\endcsname\relax\else
1394      \if@filesw
1395        \ifx\babel@aux\@gobbletwo\else % Set if single in the first, redundant
1396          % \bbl@savelastskip
1397          \protected@write\@auxout{}{\string\babel@aux{\bbl@auxname}{}}%
1398          % \bbl@restorelastskip
1399        \fi
1400        \bbl@usehooks{write}{}%
1401      \fi
1402    \fi}
1403 % The following is used above to deal with skips before the write
1404 % whatsit. Adapted from hyperref, but it might fail, so for the moment
1405 % it's not activated. TODO.
1406 \def\bbl@savelastskip{%
1407    \let\bbl@restorelastskip\relax
1408    \ifvmode
1409      \ifdim\lastskip=\z@
1410        \let\bbl@restorelastskip\nobreak
1411      \else
1412        \bbl@exp{%
1413          \def\\\bbl@restorelastskip{%
1414            \skip@=\the\lastskip
1415            \\\nobreak \vskip-\skip@ \vskip\skip@}}%
1416      \fi
1417    \fi}
1418 \newif\ifbbl@bcpallowed
1419 \bbl@bcpallowedfalse
1420 \def\select@language#1{% from set@, babel@aux
1421    % set hymap
1422    \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
1423    % set name
1424    \edef\languagename{#1}%
1425    \bbl@fixname\languagename
1426    % TODO. name@map must be here?
```

```
1427  \bbl@provide@locale
1428  \bbl@iflanguage\languagename{%
1429    \expandafter\ifx\csname date\languagename\endcsname\relax
1430    \bbl@error
1431      {Unknown language `\languagename'. Either you have\\%
1432       misspelled its name, it has not been installed,\\%
1433       or you requested it in a previous run. Fix its name,\\%
1434       install it or just rerun the file, respectively. In\\%
1435       some cases, you may need to remove the aux file}%
1436      {You may proceed, but expect wrong results}%
1437    \else
1438      % set type
1439      \let\bbl@select@type\z@
1440      \expandafter\bbl@switch\expandafter{\languagename}%
1441    \fi}}
1442 \def\babel@aux#1#2{% TODO. See how to avoid undefined nil's
1443   \select@language{#1}%
1444   \bbl@foreach\BabelContentsFiles{%
1445     \@writefile{##1}{\babel@toc{#1}{#2}}}}% %% TODO - ok in plain?
1446 \def\babel@toc#1#2{%
1447   \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.

```
1448 \newif\ifbbl@usedategroup
1449 \def\bbl@switch#1{%  from select@, foreign@
1450   % make sure there is info for the language if so requested
1451   \bbl@ensureinfo{#1}%
1452   % restore
1453   \originalTeX
1454   \expandafter\def\expandafter\originalTeX\expandafter{%
1455     \csname noextras#1\endcsname
1456     \let\originalTeX\@empty
1457     \babel@beginsave}%
1458   \bbl@usehooks{afterreset}{}%
1459   \languageshorthands{none}%
1460   % set the locale id
1461   \bbl@id@assign
1462   % switch captions, date
1463   % No text is supposed to be added here, so we remove any
1464   % spurious spaces.
1465   \bbl@bsphack
1466     \ifcase\bbl@select@type
1467       \csname captions#1\endcsname\relax
1468       \csname date#1\endcsname\relax
1469     \else
1470       \bbl@xin@{,captions,}{,\bbl@select@opts,}%
1471       \ifin@
1472         \csname captions#1\endcsname\relax
```

```
1473        \fi
1474        \bbl@xin@{,date,}{,\bbl@select@opts,}%
1475        \ifin@  % if \foreign... within \<lang>date
1476          \csname date#1\endcsname\relax
1477        \fi
1478      \fi
1479    \bbl@esphack
1480    % switch extras
1481    \bbl@usehooks{beforeextras}{}%
1482    \csname extras#1\endcsname\relax
1483    \bbl@usehooks{afterextras}{}%
1484    %  > babel-ensure
1485    %  > babel-sh-<short>
1486    %  > babel-bidi
1487    %  > babel-fontspec
1488    % hyphenation - case mapping
1489    \ifcase\bbl@opt@hyphenmap\or
1490      \def\BabelLower##1##2{\lccode##1=##2\relax}%
1491      \ifnum\bbl@hymapsel>4\else
1492        \csname\languagename @bbl@hyphenmap\endcsname
1493      \fi
1494      \chardef\bbl@opt@hyphenmap\z@
1495    \else
1496      \ifnum\bbl@hymapsel>\bbl@opt@hyphenmap\else
1497        \csname\languagename @bbl@hyphenmap\endcsname
1498      \fi
1499    \fi
1500    \let\bbl@hymapsel\@cclv
1501    % hyphenation - select rules
1502    \ifnum\csname l@\languagename\endcsname=\l@unhyphenated
1503      \edef\bbl@tempa{u}%
1504    \else
1505      \edef\bbl@tempa{\bbl@cl{lnbrk}}%
1506    \fi
1507    % linebreaking - handle u, e, k (v in the future)
1508    \bbl@xin@{/u}{/\bbl@tempa}%
1509    \ifin@\else\bbl@xin@{/e}{/\bbl@tempa}\fi % elongated forms
1510    \ifin@\else\bbl@xin@{/k}{/\bbl@tempa}\fi % only kashida
1511    \ifin@\else\bbl@xin@{/v}{/\bbl@tempa}\fi % variable font
1512    \ifin@
1513      % unhyphenated/kashida/elongated = allow stretching
1514      \language\l@unhyphenated
1515      \babel@savevariable\emergencystretch
1516      \emergencystretch\maxdimen
1517      \babel@savevariable\hbadness
1518      \hbadness\@M
1519      \hfuzz=10pt
1520    \else
1521      % other = select patterns
1522      \bbl@patterns{#1}%
1523    \fi
1524    % hyphenation - mins
1525    \babel@savevariable\lefthyphenmin
1526    \babel@savevariable\righthyphenmin
1527    \expandafter\ifx\csname #1hyphenmins\endcsname\relax
1528      \set@hyphenmins\tw@\thr@@\relax
1529    \else
1530      \expandafter\expandafter\expandafter\set@hyphenmins
1531        \csname #1hyphenmins\endcsname\relax
```

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.

```
1533 \long\def\otherlanguage#1{%
1534   \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\thr@@\fi
1535   \csname selectlanguage \endcsname{#1}%
1536   \ignorespaces}
```

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

```
1537 \long\def\endotherlanguage{%
1538   \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`.

```
1539 \expandafter\def\csname otherlanguage*\endcsname{%
1540   \@ifnextchar[\bbl@otherlanguage@s{\bbl@otherlanguage@s[]}}
1541 \def\bbl@otherlanguage@s[#1]#2{%
1542   \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
1543   \def\bbl@select@opts{#1}%
1544   \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".

```
1545 \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.

```
1546 \providecommand\bbl@beforeforeign{}
1547 \edef\foreignlanguage{%
1548   \noexpand\protect
1549   \expandafter\noexpand\csname foreignlanguage \endcsname}
1550 \expandafter\def\csname foreignlanguage \endcsname{%
```

```
1551    \@ifstar\bbl@foreign@s\bbl@foreign@x}
1552 \providecommand\bbl@foreign@x[3][]{%
1553    \begingroup
1554      \def\bbl@select@opts{#1}%
1555      \let\BabelText\@firstofone
1556      \bbl@beforeforeign
1557      \foreign@language{#2}%
1558      \bbl@usehooks{foreign}{}%
1559      \BabelText{#3}% Now in horizontal mode!
1560    \endgroup}
1561 \def\bbl@foreign@s#1#2{% TODO - \shapemode, \@setpar, ?\@@par
1562    \begingroup
1563      {\par}%
1564      \let\bbl@select@opts\@empty
1565      \let\BabelText\@firstofone
1566      \foreign@language{#1}%
1567      \bbl@usehooks{foreign*}{}%
1568      \bbl@dirparastext
1569      \BabelText{#2}% Still in vertical mode!
1570      {\par}%
1571    \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.

```
1572 \def\foreign@language#1{%
1573   % set name
1574   \edef\languagename{#1}%
1575   \ifbbl@usedategroup
1576     \bbl@add\bbl@select@opts{,date,}%
1577     \bbl@usedategroupfalse
1578   \fi
1579   \bbl@fixname\languagename
1580   % TODO. name@map here?
1581   \bbl@provide@locale
1582   \bbl@iflanguage\languagename{%
1583     \expandafter\ifx\csname date\languagename\endcsname\relax
1584       \bbl@warning   % TODO - why a warning, not an error?
1585         {Unknown language `#1'. Either you have\\%
1586          misspelled its name, it has not been installed,\\%
1587          or you requested it in a previous run. Fix its name,\\%
1588          install it or just rerun the file, respectively. In\\%
1589          some cases, you may need to remove the aux file.\\%
1590          I'll proceed, but expect wrong results.\\%
1591          Reported}%
1592     \fi
1593     % set type
1594     \let\bbl@select@type\@ne
1595     \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.

```
1596 \let\bbl@hyphlist\@empty
1597 \let\bbl@hyphenation@\relax
1598 \let\bbl@pttnlist\@empty
1599 \let\bbl@patterns@\relax
1600 \let\bbl@hymapsel=\@cclv
1601 \def\bbl@patterns#1{%
1602   \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
1603       \csname l@#1\endcsname
1604       \edef\bbl@tempa{#1}%
1605     \else
1606       \csname l@#1:\f@encoding\endcsname
1607       \edef\bbl@tempa{#1:\f@encoding}%
1608     \fi
1609   \@expandtwoargs\bbl@usehooks{patterns}{{#1}{\bbl@tempa}}%
1610   % > luatex
1611   \@ifundefined{bbl@hyphenation@}{}{% Can be \relax!
1612     \begingroup
1613       \bbl@xin@{,\number\language,}{,\bbl@hyphlist}%
1614       \ifin@\else
1615         \@expandtwoargs\bbl@usehooks{hyphenation}{{#1}{\bbl@tempa}}%
1616         \hyphenation{%
1617           \bbl@hyphenation@
1618           \@ifundefined{bbl@hyphenation@#1}%
1619             \@empty
1620             {\space\csname bbl@hyphenation@#1\endcsname}}%
1621         \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
1622       \fi
1623     \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*.

```
1624 \def\hyphenrules#1{%
1625   \edef\bbl@tempf{#1}%
1626   \bbl@fixname\bbl@tempf
1627   \bbl@iflanguage\bbl@tempf{%
1628     \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
1629     \ifx\languageshorthands\@undefined\else
1630       \languageshorthands{none}%
1631     \fi
1632     \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax
1633       \set@hyphenmins\tw@\thr@@\relax
1634     \else
1635       \expandafter\expandafter\expandafter\set@hyphenmins
1636       \csname\bbl@tempf hyphenmins\endcsname\relax
1637     \fi}}
1638 \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.

```
1639 \def\providehyphenmins#1#2{%
1640   \expandafter\ifx\csname #1hyphenmins\endcsname\relax
1641     \@namedef{#1hyphenmins}{#2}%
1642   \fi}
```

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

```
1643 \def\set@hyphenmins#1#2{%
1644   \lefthyphenmin#1\relax
1645   \righthyphenmin#2\relax}
```

\ProvidesLanguage The identification code for each file is something that was introduced in LaTeX2ε. 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.

```
1646 \ifx\ProvidesFile\@undefined
1647   \def\ProvidesLanguage#1[#2 #3 #4]{%
1648     \wlog{Language: #1 #4 #3 <#2>}%
1649     }
1650 \else
1651   \def\ProvidesLanguage#1{%
1652     \begingroup
1653       \catcode`\ 10 %
1654       \@makeother\/%
1655       \@ifnextchar[%]
1656         {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}}
1657   \def\@provideslanguage#1[#2]{%
1658     \wlog{Language: #1 #2}%
1659     \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
1660     \endgroup}
1661 \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.

```
1662 \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.

```
1663 \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':

```
1664 \providecommand\setlocale{%
1665   \bbl@error
1666     {Not yet available}%
1667     {Find an armchair, sit down and wait}}
1668 \let\uselocale\setlocale
1669 \let\locale\setlocale
1670 \let\selectlocale\setlocale
1671 \let\localename\setlocale
1672 \let\textlocale\setlocale
1673 \let\textlanguage\setlocale
1674 \let\languagetext\setlocale
```

9.2 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 LaTeX2ε, so we can safely use its error
handling interface. Otherwise we'll have to 'keep it simple'.

101

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.

```
1675 \edef\bbl@nulllanguage{\string\language=0}
1676 \ifx\PackageError\@undefined  % TODO. Move to Plain
1677   \def\bbl@error#1#2{%
1678     \begingroup
1679       \newlinechar=`\^^J
1680       \def\\{^^J(babel) }%
1681       \errhelp{#2}\errmessage{\\#1}%
1682     \endgroup}
1683   \def\bbl@warning#1{%
1684     \begingroup
1685       \newlinechar=`\^^J
1686       \def\\{^^J(babel) }%
1687       \message{\\#1}%
1688     \endgroup}
1689   \let\bbl@infowarn\bbl@warning
1690   \def\bbl@info#1{%
1691     \begingroup
1692       \newlinechar=`\^^J
1693       \def\\{^^J}%
1694       \wlog{#1}%
1695     \endgroup}
1696 \fi
1697 \def\bbl@nocaption{\protect\bbl@nocaption@i}
1698 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
1699   \global\@namedef{#2}{\textbf{?#1?}}%
1700   \@nameuse{#2}%
1701   \edef\bbl@tempa{#1}%
1702   \bbl@sreplace\bbl@tempa{name}{}%
1703   \bbl@warning{% TODO.
1704     \@backslashchar#1 not set for '\languagename'. Please,\\%
1705     define it after the language has been loaded\\%
1706     (typically in the preamble) with:\\%
1707     \string\setlocalecaption{\languagename}{\bbl@tempa}{..}\\%
1708     Reported}}
1709 \def\bbl@tentative{\protect\bbl@tentative@i}
1710 \def\bbl@tentative@i#1{%
1711   \bbl@warning{%
1712     Some functions for '#1' are tentative.\\%
1713     They might not work as expected and their behavior\\%
1714     could change in the future.\\%
1715     Reported}}
1716 \def\@nolanerr#1{%
1717   \bbl@error
1718     {You haven't defined the language #1\space yet.\\%
1719      Perhaps you misspelled it or your installation\\%
1720      is not complete}%
1721     {Your command will be ignored, type <return> to proceed}}
1722 \def\@nopatterns#1{%
1723   \bbl@warning
1724     {No hyphenation patterns were preloaded for\\%
1725      the language `#1' into the format.\\%
1726      Please, configure your TeX system to add them and\\%
1727      rebuild the format. Now I will use the patterns\\%
1728      preloaded for \bbl@nulllanguage\space instead}}
1729 \let\bbl@usehooks\@gobbletwo
1730 \ifx\bbl@onlyswitch\@empty\endinput\fi
```

```
1731  % Here ended switch.def
```

Here ended switch.def.

```
1732 \ifx\directlua\@undefined\else
1733   \ifx\bbl@luapatterns\@undefined
1734     \input luababel.def
1735   \fi
1736 \fi
1737 ⟨⟨Basic macros⟩⟩
1738 \bbl@trace{Compatibility with language.def}
1739 \ifx\bbl@languages\@undefined
1740   \ifx\directlua\@undefined
1741     \openin1 = language.def % TODO. Remove hardcoded number
1742     \ifeof1
1743       \closein1
1744       \message{I couldn't find the file language.def}
1745     \else
1746       \closein1
1747       \begingroup
1748         \def\addlanguage#1#2#3#4#5{%
1749           \expandafter\ifx\csname lang@#1\endcsname\relax\else
1750             \global\expandafter\let\csname l@#1\expandafter\endcsname
1751               \csname lang@#1\endcsname
1752           \fi}%
1753         \def\uselanguage#1{}%
1754         \input language.def
1755       \endgroup
1756     \fi
1757   \fi
1758   \chardef\l@english\z@
1759 \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.

```
1760 \def\addto#1#2{%
1761   \ifx#1\@undefined
1762     \def#1{#2}%
1763   \else
1764     \ifx#1\relax
1765       \def#1{#2}%
1766     \else
1767       {\toks@\expandafter{#1#2}%
1768         \xdef#1{\the\toks@}}%
1769     \fi
1770   \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?

```
1771 \def\bbl@withactive#1#2{%
1772   \begingroup
1773     \lccode`~=`#2\relax
1774     \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`.

```
1775 \def\bbl@redefine#1{%
1776   \edef\bbl@tempa{\bbl@stripslash#1}%
1777   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1778   \expandafter\def\csname\bbl@tempa\endcsname}
1779 \@onlypreamble\bbl@redefine
```

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

```
1780 \def\bbl@redefine@long#1{%
1781   \edef\bbl@tempa{\bbl@stripslash#1}%
1782   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1783   \expandafter\long\expandafter\def\csname\bbl@tempa\endcsname}
1784 \@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␣`.

```
1785 \def\bbl@redefinerobust#1{%
1786   \edef\bbl@tempa{\bbl@stripslash#1}%
1787   \bbl@ifunset{\bbl@tempa\space}%
1788     {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
1789      \bbl@exp{\def\\#1{\\\protect\<\bbl@tempa\space>}}}%
1790     {\bbl@exp{\let\<org@\bbl@tempa>\<\bbl@tempa\space>}}%
1791   \@namedef{\bbl@tempa\space}}
1792 \@onlypreamble\bbl@redefinerobust
```

9.3 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.

```
1793 \bbl@trace{Hooks}
1794 \newcommand\AddBabelHook[3][]{%
1795   \bbl@ifunset{bbl@hk@#2}{\EnableBabelHook{#2}}{}%
1796   \def\bbl@tempa##1,#3=##2,##3\@empty{\def\bbl@tempb{##2}}%
1797   \expandafter\bbl@tempa\bbl@evargs,#3=,\@empty
1798   \bbl@ifunset{bbl@ev@#2@#3@#1}%
1799     {\bbl@csarg\bbl@add{ev@#3@#1}{\bbl@elth{#2}}}%
1800     {\bbl@csarg\let{ev@#2@#3@#1}\relax}%
1801   \bbl@csarg\newcommand{ev@#2@#3@#1}[\bbl@tempb]}
1802 \newcommand\EnableBabelHook[1]{\bbl@csarg\let{hk@#1}\@firstofone}
1803 \newcommand\DisableBabelHook[1]{\bbl@csarg\let{hk@#1}\@gobble}
1804 \def\bbl@usehooks#1#2{%
1805   \def\bbl@elth##1{%
1806     \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#1@}#2}}%
1807   \bbl@cs{ev@#1@}%
1808   \ifx\languagename\@undefined\else % Test required for Plain (?)
1809     \def\bbl@elth##1{%
1810       \bbl@cs{hk@##1}{\bbl@cl{ev@##1@#1}#2}}%
1811     \bbl@cl{ev@#1}%
1812   \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).

```
1813 \def\bbl@evargs{,% <- don't delete this comma
1814   everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
1815   adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
1816   beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
1817   hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%
1818   beforestart=0,languagename=2}
```

\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.

```
1819 \bbl@trace{Defining babelensure}
1820 \newcommand\babelensure[2][]{%  TODO - revise test files
1821   \AddBabelHook{babel-ensure}{afterextras}{%
1822     \ifcase\bbl@select@type
1823       \bbl@cl{e}%
1824     \fi}%
1825   \begingroup
1826     \let\bbl@ens@include\@empty
1827     \let\bbl@ens@exclude\@empty
1828     \def\bbl@ens@fontenc{\relax}%
1829     \def\bbl@tempb##1{%
1830       \ifx\@empty##1\else\noexpand##1\expandafter\bbl@tempb\fi}%
1831     \edef\bbl@tempa{\bbl@tempb#1\@empty}%
1832     \def\bbl@tempb##1=##2\@@{\@namedef{bbl@ens@##1}{##2}}%
1833     \bbl@foreach\bbl@tempa{\bbl@tempb##1\@@}%
1834     \def\bbl@tempc{\bbl@ensure}%
1835     \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1836       \expandafter{\bbl@ens@include}}%
1837     \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1838       \expandafter{\bbl@ens@exclude}}%
1839     \toks@\expandafter{\bbl@tempc}%
1840     \bbl@exp{%
1841   \endgroup
1842   \def\<bbl@e@#2>{\the\toks@{\bbl@ens@fontenc}}}}}
1843 \def\bbl@ensure#1#2#3{% 1: include 2: exclude 3: fontenc
1844   \def\bbl@tempb##1{% elt for (excluding) \bbl@captionslist list
1845     \ifx##1\@undefined % 3.32 - Don't assume the macro exists
1846       \edef##1{\noexpand\bbl@nocaption
1847         {\bbl@stripslash##1}{\languagename\bbl@stripslash##1}}%
1848     \fi
1849     \ifx##1\@empty\else
1850       \in@{##1}{#2}%
1851       \ifin@\else
1852         \bbl@ifunset{bbl@ensure@\languagename}%
1853           {\bbl@exp{%
1854             \\\DeclareRobustCommand\<bbl@ensure@\languagename>[1]{%
1855               \\\foreignlanguage{\languagename}%
1856               {\ifx\relax#3\else
1857                 \\\fontencoding{#3}\\\selectfont
1858               \fi
1859               ########1}}}}%
1860           {}%
1861       \toks@\expandafter{##1}%
```

105

```
1862        \edef##1{%
1863            \bbl@csarg\noexpand{ensure@\languagename}%
1864            {\the\toks@}}%
1865        \fi
1866        \expandafter\bbl@tempb
1867      \fi}%
1868    \expandafter\bbl@tempb\bbl@captionslist\today\@empty
1869    \def\bbl@tempa##1{% elt for include list
1870      \ifx##1\@empty\else
1871        \bbl@csarg\in@{ensure@\languagename\expandafter}\expandafter{##1}%
1872        \ifin@\else
1873          \bbl@tempb##1\@empty
1874        \fi
1875        \expandafter\bbl@tempa
1876      \fi}%
1877    \bbl@tempa#1\@empty}
1878 \def\bbl@captionslist{%
1879    \prefacename\refname\abstractname\bibname\chaptername\appendixname
1880    \contentsname\listfigurename\listtablename\indexname\figurename
1881    \tablename\partname\enclname\ccname\headtoname\pagename\seename
1882    \alsoname\proofname\glossaryname}
```

9.4   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.

```
1883 \bbl@trace{Macros for setting language files up}
1884 \def\bbl@ldfinit{%
1885    \let\bbl@screset\@empty
1886    \let\BabelStrings\bbl@opt@string
1887    \let\BabelOptions\@empty
1888    \let\BabelLanguages\relax
1889    \ifx\originalTeX\@undefined
1890      \let\originalTeX\@empty
1891    \else
1892      \originalTeX
1893    \fi}
1894 \def\LdfInit#1#2{%
1895    \chardef\atcatcode=\catcode`\@
1896    \catcode`\@=11\relax
1897    \chardef\eqcatcode=\catcode`\=
1898    \catcode`\==12\relax
```

```
1899    \expandafter\if\expandafter\@backslashchar
1900                    \expandafter\@car\string#2\@nil
1901      \ifx#2\@undefined\else
1902        \ldf@quit{#1}%
1903      \fi
1904    \else
1905      \expandafter\ifx\csname#2\endcsname\relax\else
1906        \ldf@quit{#1}%
1907      \fi
1908    \fi
1909    \bbl@ldfinit}
```

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

```
1910 \def\ldf@quit#1{%
1911    \expandafter\main@language\expandafter{#1}%
1912    \catcode`\@=\atcatcode \let\atcatcode\relax
1913    \catcode`\==\eqcatcode \let\eqcatcode\relax
1914    \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.

```
1915 \def\bbl@afterldf#1{% TODO. Merge into the next macro? Unused elsewhere
1916    \bbl@afterlang
1917    \let\bbl@afterlang\relax
1918    \let\BabelModifiers\relax
1919    \let\bbl@screset\relax}%
1920 \def\ldf@finish#1{%
1921    \ifx\loadlocalcfg\@undefined\else % For LaTeX 209
1922      \loadlocalcfg{#1}%
1923    \fi
1924    \bbl@afterldf{#1}%
1925    \expandafter\main@language\expandafter{#1}%
1926    \catcode`\@=\atcatcode \let\atcatcode\relax
1927    \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.

```
1928 \@onlypreamble\LdfInit
1929 \@onlypreamble\ldf@quit
1930 \@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.

```
1931 \def\main@language#1{%
1932    \def\bbl@main@language{#1}%
1933    \let\languagename\bbl@main@language % TODO. Set localename
1934    \bbl@id@assign
1935    \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.

```
1936 \def\bbl@beforestart{%
1937    \bbl@usehooks{beforestart}{}%
1938    \global\let\bbl@beforestart\relax}
```

107

```
1939 \AtBeginDocument{%
1940   \@nameuse{bbl@beforestart}%
1941   \if@filesw
1942     \providecommand\babel@aux[2]{}%
1943     \immediate\write\@mainaux{%
1944       \string\providecommand\string\babel@aux[2]{}}%
1945     \immediate\write\@mainaux{\string\@nameuse{bbl@beforestart}}%
1946   \fi
1947   \expandafter\selectlanguage\expandafter{\bbl@main@language}%
1948   \ifbbl@single  % must go after the line above.
1949     \renewcommand\selectlanguage[1]{}%
1950     \renewcommand\foreignlanguage[2]{#2}%
1951     \global\let\babel@aux\@gobbletwo  % Also as flag
1952   \fi
1953   \ifcase\bbl@engine\or\pagedir\bodydir\fi} % TODO - a better place
```

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

```
1954 \def\select@language@x#1{%
1955   \ifcase\bbl@select@type
1956     \bbl@ifsamestring\languagename{#1}{}{\select@language{#1}}%
1957   \else
1958     \select@language{#1}%
1959   \fi}
```

9.5   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.

```
1960 \bbl@trace{Shorhands}
1961 \def\bbl@add@special#1{% 1:a macro like \", \?, etc.
1962   \bbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
1963   \bbl@ifunset{@sanitize}{}{\bbl@add\@sanitize{\@makeother#1}}%
1964   \ifx\nfss@catcodes\@undefined\else % TODO - same for above
1965     \begingroup
1966       \catcode`#1\active
1967       \nfss@catcodes
1968       \ifnum\catcode`#1=\active
1969         \endgroup
1970         \bbl@add\nfss@catcodes{\@makeother#1}%
1971       \else
1972         \endgroup
1973       \fi
1974   \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.

```
1975 \def\bbl@remove@special#1{%
1976   \begingroup
1977     \def\x##1##2{\ifnum`#1=`##2\noexpand\@empty
1978                 \else\noexpand##1\noexpand##2\fi}%
1979     \def\do{\x\do}%
1980     \def\@makeother{\x\@makeother}%
1981     \edef\x{\endgroup
1982       \def\noexpand\dospecials{\dospecials}%
```

```
1983        \expandafter\ifx\csname @sanitize\endcsname\relax\else
1984          \def\noexpand\@sanitize{\@sanitize}%
1985        \fi}%
1986    \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).

```
1987 \def\bbl@active@def#1#2#3#4{%
1988    \@namedef{#3#1}{%
1989       \expandafter\ifx\csname#2@sh@#1@\endcsname\relax
1990          \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
1991       \else
1992          \bbl@afterfi\csname#2@sh@#1@\endcsname
1993       \fi}%
```

When there is also no current-level shorthand with an argument we will check whether there is a
next-level defined shorthand for this active character.

```
1994    \long\@namedef{#3@arg#1}##1{%
1995       \expandafter\ifx\csname#2@sh@#1@\string##1@\endcsname\relax
1996          \bbl@afterelse\csname#4#1\endcsname##1%
1997       \else
1998          \bbl@afterfi\csname#2@sh@#1@\string##1@\endcsname
1999       \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.

```
2000 \def\initiate@active@char#1{%
2001    \bbl@ifunset{active@char\string#1}%
2002       {\bbl@withactive
2003          {\expandafter\@initiate@active@char\expandafter}#1\string#1#1}%
2004       {}}
```

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).

```
2005 \def\@initiate@active@char#1#2#3{%
2006    \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
2007    \ifx#1\@undefined
2008       \bbl@csarg\edef{oridef@#2}{\let\noexpand#1\noexpand\@undefined}%
2009    \else
2010       \bbl@csarg\let{oridef@@#2}#1%
2011       \bbl@csarg\edef{oridef@#2}{%
2012          \let\noexpand#1%
```

109

```
2013        \expandafter\noexpand\csname bbl@oridef@@#2\endcsname}%
2014   \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*).

```
2015   \ifx#1#3\relax
2016     \expandafter\let\csname normal@char#2\endcsname#3%
2017   \else
2018     \bbl@info{Making #2 an active character}%
2019     \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
2020       \@namedef{normal@char#2}{%
2021         \textormath{#3}{\csname bbl@oridef@@#2\endcsname}}%
2022     \else
2023       \@namedef{normal@char#2}{#3}%
2024     \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).

```
2025     \bbl@restoreactive{#2}%
2026     \AtBeginDocument{%
2027       \catcode`#2\active
2028       \if@filesw
2029         \immediate\write\@mainaux{\catcode`\string#2\active}%
2030       \fi}%
2031     \expandafter\bbl@add@special\csname#2\endcsname
2032     \catcode`#2\active
2033   \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*⟩).

```
2034   \let\bbl@tempa\@firstoftwo
2035   \if\string^#2%
2036     \def\bbl@tempa{\noexpand\textormath}%
2037   \else
2038     \ifx\bbl@mathnormal\@undefined\else
2039       \let\bbl@tempa\bbl@mathnormal
2040     \fi
2041   \fi
2042   \expandafter\edef\csname active@char#2\endcsname{%
2043     \bbl@tempa
2044       {\noexpand\if@safe@actives
2045         \noexpand\expandafter
2046         \expandafter\noexpand\csname normal@char#2\endcsname
2047       \noexpand\else
2048         \noexpand\expandafter
2049         \expandafter\noexpand\csname bbl@doactive#2\endcsname
2050       \noexpand\fi}%
2051     {\expandafter\noexpand\csname normal@char#2\endcsname}}%
2052   \bbl@csarg\edef{doactive#2}{%
2053     \expandafter\noexpand\csname user@active#2\endcsname}%
```

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

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

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

```
2054    \bbl@csarg\edef{active@#2}{%
2055      \noexpand\active@prefix\noexpand#1%
2056      \expandafter\noexpand\csname active@char#2\endcsname}%
2057    \bbl@csarg\edef{normal@#2}{%
2058      \noexpand\active@prefix\noexpand#1%
2059      \expandafter\noexpand\csname normal@char#2\endcsname}%
2060    \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.

```
2061    \bbl@active@def#2\user@group{user@active}{language@active}%
2062    \bbl@active@def#2\language@group{language@active}{system@active}%
2063    \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.

```
2064    \expandafter\edef\csname\user@group @sh@#2@@\endcsname
2065      {\expandafter\noexpand\csname normal@char#2\endcsname}%
2066    \expandafter\edef\csname\user@group @sh@#2@\string\protect@\endcsname
2067      {\expandafter\noexpand\csname user@active#2\endcsname}%
```

Finally, a couple of special cases are taken care of. (1) If we are making the right quote (') active we need to change \pr@m@s as well. Also, make sure that a single ' in math mode 'does the right thing'. (2) If we are using the caret (^) as a shorthand character special care should be taken to make sure math still works. Therefore an extra level of expansion is introduced with a check for math mode on the upper level.

```
2068    \if\string'#2%
2069      \let\prim@s\bbl@prim@s
2070      \let\active@math@prime#1%
2071    \fi
2072    \bbl@usehooks{initiateactive}{{#1}{#2}{#3}}}
```

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

```
2073 ⟨⟨*More package options⟩⟩ ≡
2074 \DeclareOption{math=active}{}
2075 \DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
2076 ⟨⟨/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.

```
2077 \@ifpackagewith{babel}{KeepShorthandsActive}%
2078    {\let\bbl@restoreactive\@gobble}%
2079    {\def\bbl@restoreactive#1{%
2080      \bbl@exp{%
2081        \\\AfterBabelLanguage\\\CurrentOption
2082          {\catcode`#1=\the\catcode`#1\relax}%
2083        \\\AtEndOfPackage
2084          {\catcode`#1=\the\catcode`#1\relax}}}%
2085    \AtEndOfPackage{\let\bbl@restoreactive\@gobble}}
```

111

\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.

```
2086 \def\bbl@sh@select#1#2{%
2087   \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
2088     \bbl@afterelse\bbl@scndcs
2089   \else
2090     \bbl@afterfi\csname#1@sh@#2@sel\endcsname
2091   \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.

```
2092 \begingroup
2093 \bbl@ifunset{ifincsname}% TODO. Ugly. Correct?
2094   {\gdef\active@prefix#1{%
2095     \ifx\protect\@typeset@protect
2096     \else
2097       \ifx\protect\@unexpandable@protect
2098         \noexpand#1%
2099       \else
2100         \protect#1%
2101       \fi
2102       \expandafter\@gobble
2103     \fi}}
2104   {\gdef\active@prefix#1{%
2105     \ifincsname
2106       \string#1%
2107       \expandafter\@gobble
2108     \else
2109       \ifx\protect\@typeset@protect
2110       \else
2111         \ifx\protect\@unexpandable@protect
2112           \noexpand#1%
2113         \else
2114           \protect#1%
2115         \fi
2116         \expandafter\expandafter\expandafter\@gobble
2117       \fi
2118     \fi}}
2119 \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*⟩.

```
2120 \newif\if@safe@actives
2121 \@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.

```
2122 \def\bbl@restore@actives{\if@safe@actives\@safe@activesfalse\fi}
```

<table>
<tr><td>\bbl@activate</td><td rowspan="2">Both macros take one argument, like \initiate@active@char. The macro is used to change the definition of an active character to expand to \active@char⟨char⟩ in the case of \bbl@activate, or \normal@char⟨char⟩ in the case of \bbl@deactivate.</td></tr>
<tr><td>\bbl@deactivate</td></tr>
</table>

```
2123 \chardef\bbl@activated\z@
2124 \def\bbl@activate#1{%
2125   \chardef\bbl@activated\@ne
2126   \bbl@withactive{\expandafter\let\expandafter}#1%
2127     \csname bbl@active@\string#1\endcsname}
2128 \def\bbl@deactivate#1{%
2129   \chardef\bbl@activated\tw@
2130   \bbl@withactive{\expandafter\let\expandafter}#1%
2131     \csname bbl@normal@\string#1\endcsname}
```

<table>
<tr><td>\bbl@firstcs</td><td rowspan="2">These macros are used only as a trick when declaring shorthands.</td></tr>
<tr><td>\bbl@scndcs</td></tr>
</table>

```
2132 \def\bbl@firstcs#1#2{\csname#1\endcsname}
2133 \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.

```
2134 \def\babel@texpdf#1#2#3#4{%
2135   \ifx\texorpdfstring\@undefined
2136     \textormath{#1}{#2}%
2137   \else
2138     \texorpdfstring{\textormath{#1}{#3}}{#2}%
2139   % \texorpdfstring{\textormath{#1}{#3}}{\textormath{#2}{#4}}%
2140   \fi}
2141 %
2142 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
2143 \def\@decl@short#1#2#3\@nil#4{%
2144   \def\bbl@tempa{#3}%
2145   \ifx\bbl@tempa\@empty
2146     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
2147     \bbl@ifunset{#1@sh@\string#2@}{}%
2148       {\def\bbl@tempa{#4}%
2149        \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbl@tempa
2150        \else
2151          \bbl@info
2152            {Redefining #1 shorthand \string#2\\%
2153             in language \CurrentOption}%
2154        \fi}%
2155     \@namedef{#1@sh@\string#2@}{#4}%
2156   \else
2157     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs
2158     \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%
2159       {\def\bbl@tempa{#4}%
2160        \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
2161        \else
2162          \bbl@info
2163            {Redefining #1 shorthand \string#2\string#3\\%
```

```
2164              in language \CurrentOption}%
2165          \fi}%
2166      \@namedef{#1@sh@\string#2@\string#3@}{#4}%
2167    \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.

```
2168 \def\textormath{%
2169    \ifmmode
2170       \expandafter\@secondoftwo
2171    \else
2172       \expandafter\@firstoftwo
2173    \fi}
```

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

```
2174 \def\user@group{user}
2175 \def\language@group{english} % TODO. I don't like defaults
2176 \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.

```
2177 \def\useshorthands{%
2178    \@ifstar\bbl@usesh@s{\bbl@usesh@x{}}}
2179 \def\bbl@usesh@s#1{%
2180    \bbl@usesh@x
2181       {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bbl@activate{#1}}}%
2182       {#1}}
2183 \def\bbl@usesh@x#1#2{%
2184    \bbl@ifshorthand{#2}%
2185       {\def\user@group{user}%
2186        \initiate@active@char{#2}%
2187        #1%
2188        \bbl@activate{#2}}%
2189       {\bbl@error
2190          {Cannot declare a shorthand turned off (\string#2)}
2191          {Sorry, but you cannot use shorthands which have been\\%
2192           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.

```
2193 \def\user@language@group{user@\language@group}
2194 \def\bbl@set@user@generic#1#2{%
2195    \bbl@ifunset{user@generic@active#1}%
2196       {\bbl@active@def#1\user@language@group{user@active}{user@generic@active}%
2197        \bbl@active@def#1\user@group{user@generic@active}{language@active}%
2198        \expandafter\edef\csname#2@sh@#1@@\endcsname{%
2199           \expandafter\noexpand\csname normal@char#1\endcsname}%
2200        \expandafter\edef\csname#2@sh@#1@\string\protect@\endcsname{%
2201           \expandafter\noexpand\csname user@active#1\endcsname}}%
2202    \@empty}
2203 \newcommand\defineshorthand[3][user]{%
2204    \edef\bbl@tempa{\zap@space#1 \@empty}%
```

```
2205    \bbl@for\bbl@tempb\bbl@tempa{%
2206      \if*\expandafter\@car\bbl@tempb\@nil
2207        \edef\bbl@tempb{user@\expandafter\@gobble\bbl@tempb}%
2208        \@expandtwoargs
2209          \bbl@set@user@generic{\expandafter\string\@car#2\@nil}\bbl@tempb
2210      \fi
2211      \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].

```
2212 \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".

```
2213 \def\aliasshorthand#1#2{%
2214   \bbl@ifshorthand{#2}%
2215     {\expandafter\ifx\csname active@char\string#2\endcsname\relax
2216        \ifx\document\@notprerr
2217          \@notshorthand{#2}%
2218        \else
2219          \initiate@active@char{#2}%
2220          \expandafter\let\csname active@char\string#2\expandafter\endcsname
2221            \csname active@char\string#1\endcsname
2222          \expandafter\let\csname normal@char\string#2\expandafter\endcsname
2223            \csname normal@char\string#1\endcsname
2224          \bbl@activate{#2}%
2225        \fi
2226      \fi}%
2227     {\bbl@error
2228        {Cannot declare a shorthand turned off (\string#2)}
2229        {Sorry, but you cannot use shorthands which have been\\%
2230         turned off in the package options}}}
```

\@notshorthand

```
2231 \def\@notshorthand#1{%
2232   \bbl@error{%
2233     The character `\string #1' should be made a shorthand character;\\%
2234     add the command \string\useshorthands\string{#1\string} to
2235     the preamble.\\%
2236     I will ignore your instruction}%
2237     {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.

```
2238 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
2239 \DeclareRobustCommand*\shorthandoff{%
2240   \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
2241 \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.

```
2242 \def\bbl@switch@sh#1#2{%
2243   \ifx#2\@nnil\else
2244     \bbl@ifunset{bbl@active@\string#2}%
2245       {\bbl@error
2246         {I cannot switch `\string#2' on or off--not a shorthand}%
2247         {This character is not a shorthand. Maybe you made\\%
2248          a typing mistake? I will ignore your instruction.}}%
2249       {\ifcase#1%    off, on, off*
2250         \catcode`#212\relax
2251        \or
2252         \catcode`#2\active
2253         \bbl@ifunset{bbl@shdef@\string#2}%
2254           {}%
2255           {\bbl@withactive{\expandafter\let\expandafter}#2%
2256             \csname bbl@shdef@\string#2\endcsname
2257            \bbl@csarg\let{shdef@\string#2}\relax}%
2258         \ifcase\bbl@activated\or
2259           \bbl@activate{#2}%
2260         \else
2261           \bbl@deactivate{#2}%
2262         \fi
2263        \or
2264         \bbl@ifunset{bbl@shdef@\string#2}%
2265           {\bbl@withactive{\bbl@csarg\let{shdef@\string#2}}#2}%
2266           {}%
2267         \csname bbl@oricat@\string#2\endcsname
2268         \csname bbl@oridef@\string#2\endcsname
2269       \fi}%
2270     \bbl@afterfi\bbl@switch@sh#1%
2271   \fi}
```

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

```
2272 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
2273 \def\bbl@putsh#1{%
2274   \bbl@ifunset{bbl@active@\string#1}%
2275     {\bbl@putsh@i#1\@empty\@nnil}%
2276     {\csname bbl@active@\string#1\endcsname}}
2277 \def\bbl@putsh@i#1#2\@nnil{%
2278   \csname\language@group @sh@\string#1@%
2279     \ifx\@empty#2\else\string#2@\fi\endcsname}
2280 \ifx\bbl@opt@shorthands\@nnil\else
2281   \let\bbl@s@initiate@active@char\initiate@active@char
2282   \def\initiate@active@char#1{%
2283     \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
2284   \let\bbl@s@switch@sh\bbl@switch@sh
2285   \def\bbl@switch@sh#1#2{%
2286     \ifx#2\@nnil\else
2287       \bbl@afterfi
2288       \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
2289     \fi}
2290   \let\bbl@s@activate\bbl@activate
2291   \def\bbl@activate#1{%
2292     \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
2293   \let\bbl@s@deactivate\bbl@deactivate
2294   \def\bbl@deactivate#1{%
2295     \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
2296 \fi
```

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

or off.

```
2297 \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.

```
2298 \def\bbl@prim@s{%
2299   \prime\futurelet\@let@token\bbl@pr@m@s}
2300 \def\bbl@if@primes#1#2{%
2301   \ifx#1\@let@token
2302     \expandafter\@firstoftwo
2303   \else\ifx#2\@let@token
2304     \bbl@afterelse\expandafter\@firstoftwo
2305   \else
2306     \bbl@afterfi\expandafter\@secondoftwo
2307   \fi\fi}
2308 \begingroup
2309   \catcode`\^=7  \catcode`\*=\active  \lccode`\*=`\^
2310   \catcode`\'=12 \catcode`\"=\active  \lccode`\"=`\'
2311   \lowercase{%
2312     \gdef\bbl@pr@m@s{%
2313       \bbl@if@primes"'%
2314         \pr@@@s
2315         {\bbl@if@primes*^\pr@@@t\egroup}}}
2316 \endgroup
```

Usually the ~ is active and expands to \penalty\@M\␣. When it is written to the .aux file it is written
expanded. To prevent that and to be able to use the character ~ as a start character for a shorthand,
it is redefined here as a one character shorthand on system level. The system declaration is in most
cases redundant (when ~ is still a non-break space), and in some cases is inconvenient (if ~ has been
redefined); however, for backward compatibility it is maintained (some existing documents may rely
on the babel value).

```
2317 \initiate@active@char{~}
2318 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
2319 \bbl@activate{~}
```

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

```
2320 \expandafter\def\csname OT1dqpos\endcsname{127}
2321 \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

```
2322 \ifx\f@encoding\@undefined
2323   \def\f@encoding{OT1}
2324 \fi
```

9.6   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.

```
2325 \bbl@trace{Language attributes}
2326 \newcommand\languageattribute[2]{%
```

```
2327    \def\bbl@tempc{#1}%
2328    \bbl@fixname\bbl@tempc
2329    \bbl@iflanguage\bbl@tempc{%
2330      \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.

```
2331        \ifx\bbl@known@attribs\@undefined
2332          \in@false
2333        \else
2334          \bbl@xin@{,\bbl@tempc-##1,}{,\bbl@known@attribs,}%
2335        \fi
2336        \ifin@
2337          \bbl@warning{%
2338            You have more than once selected the attribute '##1'\\%
2339            for language #1. Reported}%
2340        \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.

```
2341          \bbl@exp{%
2342            \\\bbl@add@list\\\bbl@known@attribs{\bbl@tempc-##1}}%
2343          \edef\bbl@tempa{\bbl@tempc-##1}%
2344          \expandafter\bbl@ifknown@ttrib\expandafter{\bbl@tempa}\bbl@attributes%
2345          {\csname\bbl@tempc @attr@##1\endcsname}%
2346          {\@attrerr{\bbl@tempc}{##1}}%
2347      \fi}}}
2348  \@onlypreamble\languageattribute
```

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

```
2349  \newcommand*{\@attrerr}[2]{%
2350    \bbl@error
2351      {The attribute #2 is unknown for language #1.}%
2352      {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}.

```
2353  \def\bbl@declare@ttribute#1#2#3{%
2354    \bbl@xin@{,#2,}{,\BabelModifiers,}%
2355    \ifin@
2356      \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
2357    \fi
2358    \bbl@add@list\bbl@attributes{#1-#2}%
2359    \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.

```
2360  \def\bbl@ifattributeset#1#2#3#4{%
2361    \ifx\bbl@known@attribs\@undefined
2362      \in@false
2363    \else
2364      \bbl@xin@{,#1-#2,}{,\bbl@known@attribs,}%
2365    \fi
```

```
2366    \ifin@
2367      \bbl@afterelse#3%
2368    \else
2369      \bbl@afterfi#4%
2370    \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.

```
2371 \def\bbl@ifknown@ttrib#1#2{%
2372    \let\bbl@tempa\@secondoftwo
2373    \bbl@loopx\bbl@tempb{#2}{%
2374      \expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{,#1,}%
2375      \ifin@
2376        \let\bbl@tempa\@firstoftwo
2377      \else
2378      \fi}%
2379    \bbl@tempa}
```

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

```
2380 \def\bbl@clear@ttribs{%
2381    \ifx\bbl@attributes\@undefined\else
2382      \bbl@loopx\bbl@tempa{\bbl@attributes}{%
2383        \expandafter\bbl@clear@ttrib\bbl@tempa.
2384        }%
2385      \let\bbl@attributes\@undefined
2386    \fi}
2387 \def\bbl@clear@ttrib#1-#2.{%
2388    \expandafter\let\csname#1@attr@#2\endcsname\@undefined}
2389 \AtBeginDocument{\bbl@clear@ttribs}
```

9.7 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
```
2390 \bbl@trace{Macros for saving definitions}
2391 \def\babel@beginsave{\babel@savecnt\z@}
```

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

```
2392 \newcount\babel@savecnt
2393 \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.

---

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

```
2394 \def\babel@save#1{%
2395   \expandafter\let\csname babel@\number\babel@savecnt\endcsname#1\relax
2396   \toks@\expandafter{\originalTeX\let#1=}%
2397   \bbl@exp{%
2398     \def\\\originalTeX{\the\toks@\<babel@\number\babel@savecnt>\relax}}%
2399   \advance\babel@savecnt\@ne}
2400 \def\babel@savevariable#1{%
2401   \toks@\expandafter{\originalTeX #1=}%
2402   \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.

```
2403 \def\bbl@frenchspacing{%
2404   \ifnum\the\sfcode`\.=\@m
2405     \let\bbl@nonfrenchspacing\relax
2406   \else
2407     \frenchspacing
2408     \let\bbl@nonfrenchspacing\nonfrenchspacing
2409   \fi}
2410 \let\bbl@nonfrenchspacing\nonfrenchspacing
2411 \let\bbl@elt\relax
2412 \edef\bbl@fs@chars{%
2413   \bbl@elt{\string.}\@m{3000}\bbl@elt{\string?}\@m{3000}%
2414   \bbl@elt{\string!}\@m{3000}\bbl@elt{\string:}\@m{2000}%
2415   \bbl@elt{\string;}\@m{1500}\bbl@elt{\string,}\@m{1250}}
```

9.8   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.

```
2416 \bbl@trace{Short tags}
2417 \def\babeltags#1{%
2418   \edef\bbl@tempa{\zap@space#1 \@empty}%
2419   \def\bbl@tempb##1=##2\@@{%
2420     \edef\bbl@tempc{%
2421       \noexpand\newcommand
2422       \expandafter\noexpand\csname ##1\endcsname{%
2423         \noexpand\protect
2424         \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}%
2425       \noexpand\newcommand
2426       \expandafter\noexpand\csname text##1\endcsname{%
2427         \noexpand\foreignlanguage{##2}}}%
2428     \bbl@tempc}%
2429   \bbl@for\bbl@tempa\bbl@tempa{%
2430     \expandafter\bbl@tempb\bbl@tempa\@@}}
```

9.9   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.

```
2431 \bbl@trace{Hyphens}
2432 \@onlypreamble\babelhyphenation
2433 \AtEndOfPackage{%
```

```
2434    \newcommand\babelhyphenation[2][\@empty]{%
2435      \ifx\bbl@hyphenation@\relax
2436        \let\bbl@hyphenation@\@empty
2437      \fi
2438      \ifx\bbl@hyphlist\@empty\else
2439        \bbl@warning{%
2440          You must not intermingle \string\selectlanguage\space and\\%
2441          \string\babelhyphenation\space or some exceptions will not\\%
2442          be taken into account. Reported}%
2443      \fi
2444      \ifx\@empty#1%
2445        \protected@edef\bbl@hyphenation@{\bbl@hyphenation@\space#2}%
2446      \else
2447        \bbl@vforeach{#1}{%
2448          \def\bbl@tempa{##1}%
2449          \bbl@fixname\bbl@tempa
2450          \bbl@iflanguage\bbl@tempa{%
2451            \bbl@csarg\protected@edef{hyphenation@\bbl@tempa}{%
2452              \bbl@ifunset{bbl@hyphenation@\bbl@tempa}%
2453                {}%
2454                {\csname bbl@hyphenation@\bbl@tempa\endcsname\space}%
2455              #2}}}%
2456      \fi}}
```

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

```
2457 \def\bbl@allowhyphens{\ifvmode\else\nobreak\hskip\z@skip\fi}
2458 \def\bbl@t@one{T1}
2459 \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.

```
2460 \newcommand\babelnullhyphen{\char\hyphenchar\font}
2461 \def\babelhyphen{\active@prefix\babelhyphen\bbl@hyphen}
2462 \def\bbl@hyphen{%
2463   \@ifstar{\bbl@hyphen@i @}{\bbl@hyphen@i\@empty}}
2464 \def\bbl@hyphen@i#1#2{%
2465   \bbl@ifunset{bbl@hy@#1#2\@empty}%
2466     {\csname bbl@#1usehyphen\endcsname{\discretionary{#2}{}{#2}}}%
2467     {\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.

```
2468 \def\bbl@usehyphen#1{%
2469   \leavevmode
2470   \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
2471   \nobreak\hskip\z@skip}
2472 \def\bbl@@usehyphen#1{%
2473   \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}
```

The following macro inserts the hyphen char.

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

```
2474 \def\bbl@hyphenchar{%
2475   \ifnum\hyphenchar\font=\m@ne
2476     \babelnullhyphen
2477   \else
2478     \char\hyphenchar\font
2479   \fi}
```

Finally, we define the hyphen "types". Their names will not change, so you may use them in ldf's. After a space, the \mbox in \bbl@hy@nobreak is redundant.

```
2480 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}{}}}
2481 \def\bbl@hy@@soft{\bbl@@usehyphen{\discretionary{\bbl@hyphenchar}{}{}}}
2482 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
2483 \def\bbl@hy@@hard{\bbl@@usehyphen\bbl@hyphenchar}
2484 \def\bbl@hy@nobreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}}
2485 \def\bbl@hy@@nobreak{\mbox{\bbl@hyphenchar}}
2486 \def\bbl@hy@repeat{%
2487   \bbl@usehyphen{%
2488     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
2489 \def\bbl@hy@@repeat{%
2490   \bbl@@usehyphen{%
2491     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
2492 \def\bbl@hy@empty{\hskip\z@skip}
2493 \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.

```
2494 \def\bbl@disc#1#2{\nobreak\discretionary{#2-}{}{#1}\bbl@allowhyphens}
```

9.10 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.

```
2495 \bbl@trace{Multiencoding strings}
2496 \def\bbl@toglobal#1{\global\let#1#1}
2497 \def\bbl@recatcode#1{% TODO. Used only once?
2498   \@tempcnta="7F
2499   \def\bbl@tempa{%
2500     \ifnum\@tempcnta>"FF\else
2501       \catcode\@tempcnta=#1\relax
2502       \advance\@tempcnta\@ne
2503       \expandafter\bbl@tempa
2504     \fi}%
2505   \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).

```
2506 \@ifpackagewith{babel}{nocase}%
2507   {\let\bbl@patchuclc\relax}%
2508   {\def\bbl@patchuclc{%
2509     \global\let\bbl@patchuclc\relax
2510     \g@addto@macro\@uclclist{\reserved@b{\reserved@b\bbl@uclc}}%
2511     \gdef\bbl@uclc##1{%
2512       \let\bbl@encoded\bbl@encoded@uclc
2513       \bbl@ifunset{\languagename @bbl@uclc}% and resumes it
2514         {##1}%
2515         {\let\bbl@tempa##1\relax % Used by LANG@bbl@uclc
2516          \csname\languagename @bbl@uclc\endcsname}%
2517       {\bbl@tolower\@empty}{\bbl@toupper\@empty}}%
2518     \gdef\bbl@tolower{\csname\languagename @bbl@lc\endcsname}%
2519     \gdef\bbl@toupper{\csname\languagename @bbl@uc\endcsname}}}
```

```
2520 ⟨⟨*More package options⟩⟩ ≡
2521 \DeclareOption{nocase}{}
2522 ⟨⟨/More package options⟩⟩
```

The following package options control the behavior of \SetString.

```
2523 ⟨⟨*More package options⟩⟩ ≡
2524 \let\bbl@opt@strings\@nnil % accept strings=value
2525 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
2526 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
2527 \def\BabelStringsDefault{generic}
2528 ⟨⟨/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.

```
2529 \@onlypreamble\StartBabelCommands
2530 \def\StartBabelCommands{%
2531   \begingroup
2532   \bbl@recatcode{11}%
2533   ⟨⟨Macros local to BabelCommands⟩⟩
2534   \def\bbl@provstring##1##2{%
2535     \providecommand##1{##2}%
2536     \bbl@toglobal##1}%
2537   \global\let\bbl@scafter\@empty
2538   \let\StartBabelCommands\bbl@startcmds
2539   \ifx\BabelLanguages\relax
2540     \let\BabelLanguages\CurrentOption
2541   \fi
2542   \begingroup
2543   \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
2544   \StartBabelCommands}
2545 \def\bbl@startcmds{%
2546   \ifx\bbl@screset\@nnil\else
2547     \bbl@usehooks{stopcommands}{}%
2548   \fi
2549   \endgroup
2550   \begingroup
2551   \@ifstar
2552     {\ifx\bbl@opt@strings\@nnil
2553        \let\bbl@opt@strings\BabelStringsDefault
2554      \fi
2555      \bbl@startcmds@i}%
```

```
2556        \bbl@startcmds@i}
2557 \def\bbl@startcmds@i#1#2{%
2558    \edef\bbl@L{\zap@space#1 \@empty}%
2559    \edef\bbl@G{\zap@space#2 \@empty}%
2560    \bbl@startcmds@ii}
2561 \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.

```
2562 \newcommand\bbl@startcmds@ii[1][\@empty]{%
2563    \let\SetString\@gobbletwo
2564    \let\bbl@stringdef\@gobbletwo
2565    \let\AfterBabelCommands\@gobble
2566    \ifx\@empty#1%
2567      \def\bbl@sc@label{generic}%
2568      \def\bbl@encstring##1##2{%
2569        \ProvideTextCommandDefault##1{##2}%
2570        \bbl@toglobal##1%
2571        \expandafter\bbl@toglobal\csname\string?\string##1\endcsname}%
2572      \let\bbl@sctest\in@true
2573    \else
2574      \let\bbl@sc@charset\space % <- zapped below
2575      \let\bbl@sc@fontenc\space % <-    "        "
2576      \def\bbl@tempa##1=##2\@nil{%
2577        \bbl@csarg\edef{sc@\zap@space##1 \@empty}{##2 }}%
2578      \bbl@vforeach{label=#1}{\bbl@tempa##1\@nil}%
2579      \def\bbl@tempa##1 ##2{% space -> comma
2580        ##1%
2581        \ifx\@empty##2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa##2\fi}%
2582      \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%
2583      \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%
2584      \edef\bbl@sc@charset{\expandafter\zap@space\bbl@sc@charset\@empty}%
2585      \def\bbl@encstring##1##2{%
2586        \bbl@foreach\bbl@sc@fontenc{%
2587          \bbl@ifunset{T@####1}%
2588            {}%
2589            {\ProvideTextCommand##1{####1}{##2}%
2590             \bbl@toglobal##1%
2591             \expandafter
2592             \bbl@toglobal\csname####1\string##1\endcsname}}}%
2593      \def\bbl@sctest{%
2594        \bbl@xin@{,\bbl@opt@strings,}{,\bbl@sc@label,\bbl@sc@fontenc,}}%
2595    \fi
2596    \ifx\bbl@opt@strings\@nnil        % ie, no strings key -> defaults
2597    \else\ifx\bbl@opt@strings\relax   % ie, strings=encoded
2598      \let\AfterBabelCommands\bbl@aftercmds
2599      \let\SetString\bbl@setstring
2600      \let\bbl@stringdef\bbl@encstring
2601    \else       % ie, strings=value
2602    \bbl@sctest
2603    \ifin@
2604      \let\AfterBabelCommands\bbl@aftercmds
```

124

```
2605    \let\SetString\bbl@setstring
2606    \let\bbl@stringdef\bbl@provstring
2607 \fi\fi\fi
2608 \bbl@scswitch
2609 \ifx\bbl@G\@empty
2610    \def\SetString##1##2{%
2611      \bbl@error{Missing group for string \string##1}%
2612        {You must assign strings to some category, typically\\%
2613         captions or extras, but you set none}}%
2614 \fi
2615 \ifx\@empty#1%
2616    \bbl@usehooks{defaultcommands}{}%
2617 \else
2618    \@expandtwoargs
2619    \bbl@usehooks{encodedcommands}{{\bbl@sc@charset}{\bbl@sc@fontenc}}%
2620 \fi}
```

There are two versions of \bbl@scswitch. The first version is used when ldfs are read, and it makes sure \⟨*group*⟩⟨*language*⟩ is reset, but only once (\bbl@screset is used to keep track of this). The second version is used in the preamble and packages loaded after babel and does nothing. The macro \bbl@forlang loops \bbl@L but its body is executed only if the value is in \BabelLanguages (inside babel) or \date⟨*language*⟩ is defined (after babel has been loaded). There are also two version of \bbl@forlang. The first one skips the current iteration if the language is not in \BabelLanguages (used in ldfs), and the second one skips undefined languages (after babel has been loaded) .

```
2621 \def\bbl@forlang#1#2{%
2622   \bbl@for#1\bbl@L{%
2623     \bbl@xin@{,#1,}{,\BabelLanguages,}%
2624     \ifin@#2\relax\fi}}
2625 \def\bbl@scswitch{%
2626   \bbl@forlang\bbl@tempa{%
2627     \ifx\bbl@G\@empty\else
2628       \ifx\SetString\@gobbletwo\else
2629         \edef\bbl@GL{\bbl@G\bbl@tempa}%
2630         \bbl@xin@{,\bbl@GL,}{,\bbl@screset,}%
2631         \ifin@\else
2632           \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
2633           \xdef\bbl@screset{\bbl@screset,\bbl@GL}%
2634         \fi
2635       \fi
2636     \fi}}
2637 \AtEndOfPackage{%
2638   \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{}{#2}}}%
2639   \let\bbl@scswitch\relax}
2640 \@onlypreamble\EndBabelCommands
2641 \def\EndBabelCommands{%
2642   \bbl@usehooks{stopcommands}{}%
2643   \endgroup
2644   \endgroup
2645   \bbl@scafter}
2646 \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.

```
2647 \def\bbl@setstring#1#2{% eg, \prefacename{<string>}
2648   \bbl@forlang\bbl@tempa{%
2649     \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
2650     \bbl@ifunset{\bbl@LC}% eg, \germanchaptername
2651       {\bbl@exp{%
2652         \global\\\bbl@add\<\bbl@G\bbl@tempa>{\\\bbl@scset\\#1\<\bbl@LC>}}}%
2653       {}%
2654     \def\BabelString{#2}%
2655     \bbl@usehooks{stringprocess}{}%
2656     \expandafter\bbl@stringdef
2657       \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`.

```
2658 \ifx\bbl@opt@strings\relax
2659   \def\bbl@scset#1#2{\def#1{\bbl@encoded#2}}
2660   \bbl@patchuclc
2661   \let\bbl@encoded\relax
2662   \def\bbl@encoded@uclc#1{%
2663     \@inmathwarn#1%
2664     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
2665       \expandafter\ifx\csname ?\string#1\endcsname\relax
2666         \TextSymbolUnavailable#1%
2667       \else
2668         \csname ?\string#1\endcsname
2669       \fi
2670     \else
2671       \csname\cf@encoding\string#1\endcsname
2672     \fi}
2673 \else
2674   \def\bbl@scset#1#2{\def#1{#2}}
2675 \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.

```
2676 ⟨⟨∗Macros local to BabelCommands⟩⟩ ≡
2677 \def\SetStringLoop##1##2{%
2678   \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
2679   \count@\z@
2680   \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
2681     \advance\count@\@ne
2682     \toks@\expandafter{\bbl@tempa}%
2683     \bbl@exp{%
2684       \\\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
2685       \count@=\the\count@\relax}}%
2686 ⟨⟨/Macros local to BabelCommands⟩⟩
```

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

```
2687 \def\bbl@aftercmds#1{%
2688   \toks@\expandafter{\bbl@scafter#1}%
2689   \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.

```
2690 ⟨⟨∗Macros local to BabelCommands⟩⟩ ≡
```

```
2691  \newcommand\SetCase[3][]{%
2692    \bbl@patchuclc
2693    \bbl@forlang\bbl@tempa{%
2694      \expandafter\bbl@encstring
2695        \csname\bbl@tempa @bbl@uclc\endcsname{\bbl@tempa##1}%
2696      \expandafter\bbl@encstring
2697        \csname\bbl@tempa @bbl@uc\endcsname{##2}%
2698      \expandafter\bbl@encstring
2699        \csname\bbl@tempa @bbl@lc\endcsname{##3}}}%
2700 ⟨⟨/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.

```
2701 ⟨⟨∗Macros local to BabelCommands⟩⟩ ≡
2702    \newcommand\SetHyphenMap[1]{%
2703    \bbl@forlang\bbl@tempa{%
2704      \expandafter\bbl@stringdef
2705        \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
2706 ⟨⟨/Macros local to BabelCommands⟩⟩
```

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

```
2707 \newcommand\BabelLower[2]{% one to one.
2708    \ifnum\lccode#1=#2\else
2709      \babel@savevariable{\lccode#1}%
2710      \lccode#1=#2\relax
2711    \fi}
2712 \newcommand\BabelLowerMM[4]{% many-to-many
2713    \@tempcnta=#1\relax
2714    \@tempcntb=#4\relax
2715    \def\bbl@tempa{%
2716      \ifnum\@tempcnta>#2\else
2717        \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
2718        \advance\@tempcnta#3\relax
2719        \advance\@tempcntb#3\relax
2720        \expandafter\bbl@tempa
2721      \fi}%
2722    \bbl@tempa}
2723 \newcommand\BabelLowerMO[4]{% many-to-one
2724    \@tempcnta=#1\relax
2725    \def\bbl@tempa{%
2726      \ifnum\@tempcnta>#2\else
2727        \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
2728        \advance\@tempcnta#3
2729        \expandafter\bbl@tempa
2730      \fi}%
2731    \bbl@tempa}
```

The following package options control the behavior of hyphenation mapping.

```
2732 ⟨⟨∗More package options⟩⟩ ≡
2733 \DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\z@}
2734 \DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap\@ne}
2735 \DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\tw@}
2736 \DeclareOption{hyphenmap=other}{\chardef\bbl@opt@hyphenmap\thr@@}
2737 \DeclareOption{hyphenmap=other*}{\chardef\bbl@opt@hyphenmap4\relax}
2738 ⟨⟨/More package options⟩⟩
```

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

```
2739 \AtEndOfPackage{%
```

```
2740    \ifx\bbl@opt@hyphenmap\@undefined
2741      \bbl@xin@{,}{\bbl@language@opts}%
2742      \chardef\bbl@opt@hyphenmap\ifin@4\else\@ne\fi
2743    \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.

```
2744 \newcommand\setlocalecaption{%  TODO. Catch typos. What about ensure?
2745    \@ifstar\bbl@setcaption@s\bbl@setcaption@x}
2746 \def\bbl@setcaption@x#1#2#3{%  language caption-name string
2747    \bbl@trim@def\bbl@tempa{#2}%
2748    \bbl@xin@{.template}{\bbl@tempa}%
2749    \ifin@
2750      \bbl@ini@captions@template{#3}{#1}%
2751    \else
2752      \edef\bbl@tempd{%
2753        \expandafter\expandafter\expandafter
2754        \strip@prefix\expandafter\meaning\csname captions#1\endcsname}%
2755      \bbl@xin@
2756        {\expandafter\string\csname #2name\endcsname}%
2757        {\bbl@tempd}%
2758      \ifin@ % Renew caption
2759        \bbl@xin@{\string\bbl@scset}{\bbl@tempd}%
2760        \ifin@
2761          \bbl@exp{%
2762            \\\bbl@ifsamestring{\bbl@tempa}{\languagename}%
2763              {\\\bbl@scset\<#2name>\<#1#2name>}%
2764              {}}%
2765        \else % Old way converts to new way
2766          \bbl@ifunset{#1#2name}%
2767            {\bbl@exp{%
2768              \\\bbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}%
2769              \\\bbl@ifsamestring{\bbl@tempa}{\languagename}%
2770                {\def\<#2name>{\<#1#2name>}}%
2771                {}}}%
2772            {}%
2773        \fi
2774      \else
2775        \bbl@xin@{\string\bbl@scset}{\bbl@tempd}% New
2776        \ifin@ % New way
2777          \bbl@exp{%
2778            \\\bbl@add\<captions#1>{\\\bbl@scset\<#2name>\<#1#2name>}%
2779            \\\bbl@ifsamestring{\bbl@tempa}{\languagename}%
2780              {\\\bbl@scset\<#2name>\<#1#2name>}%
2781              {}}%
2782        \else  % Old way, but defined in the new way
2783          \bbl@exp{%
2784            \\\bbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}%
2785            \\\bbl@ifsamestring{\bbl@tempa}{\languagename}%
2786              {\def\<#2name>{\<#1#2name>}}%
2787              {}}%
2788        \fi%
2789      \fi
2790      \@namedef{#1#2name}{#3}%
2791      \toks@\expandafter{\bbl@captionslist}%
2792      \bbl@exp{\\\in@{\<#2name>}{\the\toks@}}%
2793      \ifin@\else
2794        \bbl@exp{\\\bbl@add\\\bbl@captionslist{\<#2name>}}%
```

```
2795        \bbl@toglobal\bbl@captionslist
2796      \fi
2797    \fi}
2798 % \def\bbl@setcaption@s#1#2#3{}  % TODO. Not yet implemented
```

9.11 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.

```
2799 \bbl@trace{Macros related to glyphs}
2800 \def\set@low@box#1{\setbox\tw@\hbox{,}\setbox\z@\hbox{#1}%
2801    \dimen\z@\ht\z@ \advance\dimen\z@ -\ht\tw@%
2802    \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.

```
2803 \def\save@sf@q#1{\leavevmode
2804    \begingroup
2805    \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
2806    \endgroup}
```

9.12 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.

9.12.1 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.

```
2807 \ProvideTextCommand{\quotedblbase}{OT1}{%
2808    \save@sf@q{\set@low@box{\textquotedblright\/}%
2809        \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.

```
2810 \ProvideTextCommandDefault{\quotedblbase}{%
2811    \UseTextSymbol{OT1}{\quotedblbase}}
```

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

```
2812 \ProvideTextCommand{\quotesinglbase}{OT1}{%
2813    \save@sf@q{\set@low@box{\textquoteright\/}%
2814        \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.

```
2815 \ProvideTextCommandDefault{\quotesinglbase}{%
2816    \UseTextSymbol{OT1}{\quotesinglbase}}
```

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

```
2817 \ProvideTextCommand{\guillemetleft}{OT1}{%
2818    \ifmmode
2819      \ll
2820    \else
2821      \save@sf@q{\nobreak
2822        \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2823    \fi}
2824 \ProvideTextCommand{\guillemetright}{OT1}{%
```

```
2825    \ifmmode
2826      \gg
2827    \else
2828      \save@sf@q{\nobreak
2829        \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2830    \fi}
2831 \ProvideTextCommand{\guillemotleft}{OT1}{%
2832    \ifmmode
2833      \ll
2834    \else
2835      \save@sf@q{\nobreak
2836        \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2837    \fi}
2838 \ProvideTextCommand{\guillemotright}{OT1}{%
2839    \ifmmode
2840      \gg
2841    \else
2842      \save@sf@q{\nobreak
2843        \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2844    \fi}
```

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

```
2845 \ProvideTextCommandDefault{\guillemetleft}{%
2846    \UseTextSymbol{OT1}{\guillemetleft}}
2847 \ProvideTextCommandDefault{\guillemetright}{%
2848    \UseTextSymbol{OT1}{\guillemetright}}
2849 \ProvideTextCommandDefault{\guillemotleft}{%
2850    \UseTextSymbol{OT1}{\guillemotleft}}
2851 \ProvideTextCommandDefault{\guillemotright}{%
2852    \UseTextSymbol{OT1}{\guillemotright}}
```

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

```
2853 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2854    \ifmmode
2855      <%
2856    \else
2857      \save@sf@q{\nobreak
2858        \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
2859    \fi}
2860 \ProvideTextCommand{\guilsinglright}{OT1}{%
2861    \ifmmode
2862      >%
2863    \else
2864      \save@sf@q{\nobreak
2865        \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
2866    \fi}
```

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

```
2867 \ProvideTextCommandDefault{\guilsinglleft}{%
2868    \UseTextSymbol{OT1}{\guilsinglleft}}
2869 \ProvideTextCommandDefault{\guilsinglright}{%
2870    \UseTextSymbol{OT1}{\guilsinglright}}
```

9.12.2   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.

```
2871 \DeclareTextCommand{\ij}{OT1}{%
2872    i\kern-0.02em\bbl@allowhyphens j}
```

```
2873 \DeclareTextCommand{\IJ}{OT1}{%
2874     I\kern-0.02em\bbl@allowhyphens J}
2875 \DeclareTextCommand{\ij}{T1}{\char188}
2876 \DeclareTextCommand{\IJ}{T1}{\char156}
```

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

```
2877 \ProvideTextCommandDefault{\ij}{%
2878     \UseTextSymbol{OT1}{\ij}}
2879 \ProvideTextCommandDefault{\IJ}{%
2880     \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).

```
2881 \def\crrtic@{\hrule height0.1ex width0.3em}
2882 \def\crttic@{\hrule height0.1ex width0.33em}
2883 \def\ddj@{%
2884     \setbox0\hbox{d}\dimen@=\ht0
2885     \advance\dimen@1ex
2886     \dimen@.45\dimen@
2887     \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2888     \advance\dimen@ii.5ex
2889     \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2890 \def\DDJ@{%
2891     \setbox0\hbox{D}\dimen@=.55\ht0
2892     \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2893     \advance\dimen@ii.15ex %              correction for the dash position
2894     \advance\dimen@ii-.15\fontdimen7\font %     correction for cmtt font
2895     \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
2896     \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
2897 %
2898 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2899 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}
```

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

```
2900 \ProvideTextCommandDefault{\dj}{%
2901     \UseTextSymbol{OT1}{\dj}}
2902 \ProvideTextCommandDefault{\DJ}{%
2903     \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.

```
2904 \DeclareTextCommand{\SS}{OT1}{SS}
2905 \ProvideTextCommandDefault{\SS}{\UseTextSymbol{OT1}{\SS}}
```

9.12.3 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
```
2906 \ProvideTextCommandDefault{\glq}{%
2907     \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}
```

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

```
2908 \ProvideTextCommand{\grq}{T1}{%
2909     \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}
```

131

```
2910 \ProvideTextCommand{\grq}{TU}{%
2911   \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
2912 \ProvideTextCommand{\grq}{OT1}{%
2913   \save@sf@q{\kern-.0125em
2914     \textormath{\textquoteleft}{\mbox{\textquoteleft}}%
2915     \kern.07em\relax}}
2916 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}\grq}
```

\glqq The 'german' double quotes.
\grqq
```
2917 \ProvideTextCommandDefault{\glqq}{%
2918   \textormath{\quotedblbase}{\mbox{\quotedblbase}}}
```

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

```
2919 \ProvideTextCommand{\grqq}{T1}{%
2920   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2921 \ProvideTextCommand{\grqq}{TU}{%
2922   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2923 \ProvideTextCommand{\grqq}{OT1}{%
2924   \save@sf@q{\kern-.07em
2925     \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}%
2926     \kern.07em\relax}}
2927 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}
```

\flq The 'french' single guillemets.
\frq
```
2928 \ProvideTextCommandDefault{\flq}{%
2929   \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
2930 \ProvideTextCommandDefault{\frq}{%
2931   \textormath{\guilsinglright}{\mbox{\guilsinglright}}}
```

\flqq The 'french' double guillemets.
\frqq
```
2932 \ProvideTextCommandDefault{\flqq}{%
2933   \textormath{\guillemetleft}{\mbox{\guillemetleft}}}
2934 \ProvideTextCommandDefault{\frqq}{%
2935   \textormath{\guillemetright}{\mbox{\guillemetright}}}
```

9.12.4 Umlauts and tremas

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

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

```
2936 \def\umlauthigh{%
2937   \def\bbl@umlauta##1{\leavevmode\bgroup%
2938       \expandafter\accent\csname\f@encoding dqpos\endcsname
2939       ##1\bbl@allowhyphens\egroup}%
2940   \let\bbl@umlaute\bbl@umlauta}
2941 \def\umlautlow{%
2942   \def\bbl@umlauta{\protect\lower@umlaut}}
2943 \def\umlautelow{%
2944   \def\bbl@umlaute{\protect\lower@umlaut}}
2945 \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.

```
2946 \expandafter\ifx\csname U@D\endcsname\relax
2947   \csname newdimen\endcsname\U@D
2948 \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.

```
2949 \def\lower@umlaut#1{%
2950   \leavevmode\bgroup
2951     \U@D 1ex%
2952     {\setbox\z@\hbox{%
2953       \expandafter\char\csname\f@encoding dqpos\endcsname}%
2954       \dimen@ -.45ex\advance\dimen@\ht\z@
2955       \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2956     \expandafter\accent\csname\f@encoding dqpos\endcsname
2957     \fontdimen5\font\U@D #1%
2958   \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).

```
2959 \AtBeginDocument{%
2960   \DeclareTextCompositeCommand{\"}{OT1}{a}{\bbl@umlauta{a}}%
2961   \DeclareTextCompositeCommand{\"}{OT1}{e}{\bbl@umlaute{e}}%
2962   \DeclareTextCompositeCommand{\"}{OT1}{i}{\bbl@umlaute{\i}}%
2963   \DeclareTextCompositeCommand{\"}{OT1}{\i}{\bbl@umlaute{\i}}%
2964   \DeclareTextCompositeCommand{\"}{OT1}{o}{\bbl@umlauta{o}}%
2965   \DeclareTextCompositeCommand{\"}{OT1}{u}{\bbl@umlauta{u}}%
2966   \DeclareTextCompositeCommand{\"}{OT1}{A}{\bbl@umlauta{A}}%
2967   \DeclareTextCompositeCommand{\"}{OT1}{E}{\bbl@umlaute{E}}%
2968   \DeclareTextCompositeCommand{\"}{OT1}{I}{\bbl@umlaute{I}}%
2969   \DeclareTextCompositeCommand{\"}{OT1}{O}{\bbl@umlauta{O}}%
2970   \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.

```
2971 \ifx\l@english\@undefined
2972   \chardef\l@english\z@
2973 \fi
2974 % The following is used to cancel rules in ini files (see Amharic).
2975 \ifx\l@unhyphenated\@undefined
2976   \newlanguage\l@unhyphenated
2977 \fi
```

9.13 Layout

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

```
2978 \bbl@trace{Bidi layout}
2979 \providecommand\IfBabelLayout[3]{#3}%
2980 \newcommand\BabelPatchSection[1]{%
2981   \@ifundefined{#1}{}{%
```

```
2982      \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
2983      \@namedef{#1}{%
2984        \@ifstar{\bbl@presec@s{#1}}%
2985                {\@dblarg{\bbl@presec@x{#1}}}}}}}
2986 \def\bbl@presec@x#1[#2]#3{%
2987    \bbl@exp{%
2988      \\\select@language@x{\bbl@main@language}%
2989      \\\bbl@cs{sspre@#1}%
2990      \\\bbl@cs{ss@#1}%
2991        [\\\foreignlanguage{\languagename}{\unexpanded{#2}}]%
2992        {\\\foreignlanguage{\languagename}{\unexpanded{#3}}}%
2993      \\\select@language@x{\languagename}}}
2994 \def\bbl@presec@s#1#2{%
2995    \bbl@exp{%
2996      \\\select@language@x{\bbl@main@language}%
2997      \\\bbl@cs{sspre@#1}%
2998      \\\bbl@cs{ss@#1}*%
2999        {\\\foreignlanguage{\languagename}{\unexpanded{#2}}}%
3000      \\\select@language@x{\languagename}}}
3001 \IfBabelLayout{sectioning}%
3002    {\BabelPatchSection{part}%
3003     \BabelPatchSection{chapter}%
3004     \BabelPatchSection{section}%
3005     \BabelPatchSection{subsection}%
3006     \BabelPatchSection{subsubsection}%
3007     \BabelPatchSection{paragraph}%
3008     \BabelPatchSection{subparagraph}%
3009     \def\babel@toc#1{%
3010       \select@language@x{\bbl@main@language}}}{}
3011 \IfBabelLayout{captions}%
3012    {\BabelPatchSection{caption}}{}
```

9.14 Load engine specific macros

```
3013 \bbl@trace{Input engine specific macros}
3014 \ifcase\bbl@engine
3015   \input txtbabel.def
3016 \or
3017   \input luababel.def
3018 \or
3019   \input xebabel.def
3020 \fi
```

9.15 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.

```
3021 \bbl@trace{Creating languages and reading ini files}
3022 \newcommand\babelprovide[2][]{%
3023   \let\bbl@savelangname\languagename
3024   \edef\bbl@savelocaleid{\the\localeid}%
3025   % Set name and locale id
3026   \edef\languagename{#2}%
3027   % \global\@namedef{bbl@lcname@#2}{#2}%
3028   \bbl@id@assign
3029   \let\bbl@KVP@captions\@nil
3030   \let\bbl@KVP@date\@nil
3031   \let\bbl@KVP@import\@nil
```

```
3032    \let\bbl@KVP@main\@nil
3033    \let\bbl@KVP@script\@nil
3034    \let\bbl@KVP@language\@nil
3035    \let\bbl@KVP@hyphenrules\@nil
3036    \let\bbl@KVP@linebreaking\@nil
3037    \let\bbl@KVP@mapfont\@nil
3038    \let\bbl@KVP@maparabic\@nil
3039    \let\bbl@KVP@mapdigits\@nil
3040    \let\bbl@KVP@intraspace\@nil
3041    \let\bbl@KVP@intrapenalty\@nil
3042    \let\bbl@KVP@onchar\@nil
3043    \let\bbl@KVP@transforms\@nil
3044    \global\let\bbl@release@transforms\@empty
3045    \let\bbl@KVP@alph\@nil
3046    \let\bbl@KVP@Alph\@nil
3047    \let\bbl@KVP@labels\@nil
3048    \bbl@csarg\let{KVP@labels*}\@nil
3049    \global\let\bbl@inidata\@empty
3050    \bbl@forkv{#1}{%  TODO - error handling
3051      \in@{/}{##1}%
3052      \ifin@
3053        \bbl@renewinikey##1\@@{##2}%
3054      \else
3055        \bbl@csarg\def{KVP@##1}{##2}%
3056      \fi}%
3057    % == init ==
3058    \ifx\bbl@screset\@undefined
3059      \bbl@ldfinit
3060    \fi
3061    % ==
3062    \let\bbl@lbkflag\relax % \@empty = do setup linebreak
3063    \bbl@ifunset{date#2}%
3064      {\let\bbl@lbkflag\@empty}% new
3065      {\ifx\bbl@KVP@hyphenrules\@nil\else
3066         \let\bbl@lbkflag\@empty
3067       \fi
3068       \ifx\bbl@KVP@import\@nil\else
3069         \let\bbl@lbkflag\@empty
3070       \fi}%
3071    % == import, captions ==
3072    \ifx\bbl@KVP@import\@nil\else
3073      \bbl@exp{\\\bbl@ifblank{\bbl@KVP@import}}%
3074        {\ifx\bbl@initoload\relax
3075           \begingroup
3076             \def\BabelBeforeIni##1##2{\gdef\bbl@KVP@import{##1}\endinput}%
3077             \bbl@input@texini{#2}%
3078           \endgroup
3079         \else
3080           \xdef\bbl@KVP@import{\bbl@initoload}%
3081         \fi}%
3082        {}%
3083    \fi
3084    \ifx\bbl@KVP@captions\@nil
3085      \let\bbl@KVP@captions\bbl@KVP@import
3086    \fi
3087    % ==
3088    \ifx\bbl@KVP@transforms\@nil\else
3089      \bbl@replace\bbl@KVP@transforms{ }{,}%
3090    \fi
```

```
3091  % Load ini
3092  \bbl@ifunset{date#2}%
3093    {\bbl@provide@new{#2}}%
3094    {\bbl@ifblank{#1}%
3095      {}%  With \bbl@load@basic below
3096      {\bbl@provide@renew{#2}}}%
3097  % Post tasks
3098  % ----------
3099  % == ensure captions ==
3100  \ifx\bbl@KVP@captions\@nil\else
3101    \bbl@ifunset{bbl@extracaps@#2}%
3102      {\bbl@exp{\\\babelensure[exclude=\\\today]{#2}}}%
3103      {\toks@\expandafter\expandafter\expandafter
3104        {\csname bbl@extracaps@#2\endcsname}%
3105       \bbl@exp{\\\babelensure[exclude=\\\today,include=\the\toks@]{#2}}}%
3106    \bbl@ifunset{bbl@ensure@\languagename}%
3107      {\bbl@exp{%
3108        \\\DeclareRobustCommand\<bbl@ensure@\languagename>[1]{%
3109          \\\foreignlanguage{\languagename}%
3110          {####1}}}}%
3111      {}%
3112    \bbl@exp{%
3113      \\\bbl@toglobal\<bbl@ensure@\languagename>%
3114      \\\bbl@toglobal\<bbl@ensure@\languagename\space>}%
3115  \fi
3116  % ==
3117  % At this point all parameters are defined if 'import'. Now we
3118  % execute some code depending on them. But what about if nothing was
3119  % imported? We just set the basic parameters, but still loading the
3120  % whole ini file.
3121  \bbl@load@basic{#2}%
3122  % == script, language ==
3123  % Override the values from ini or defines them
3124  \ifx\bbl@KVP@script\@nil\else
3125    \bbl@csarg\edef{sname@#2}{\bbl@KVP@script}%
3126  \fi
3127  \ifx\bbl@KVP@language\@nil\else
3128    \bbl@csarg\edef{lname@#2}{\bbl@KVP@language}%
3129  \fi
3130   % == onchar ==
3131  \ifx\bbl@KVP@onchar\@nil\else
3132    \bbl@luahyphenate
3133    \directlua{
3134      if Babel.locale_mapped == nil then
3135        Babel.locale_mapped = true
3136        Babel.linebreaking.add_before(Babel.locale_map)
3137        Babel.loc_to_scr = {}
3138        Babel.chr_to_loc = Babel.chr_to_loc or {}
3139      end}%
3140    \bbl@xin@{ ids }{ \bbl@KVP@onchar\space}%
3141    \ifin@
3142      \ifx\bbl@starthyphens\@undefined % Needed if no explicit selection
3143        \AddBabelHook{babel-onchar}{beforestart}{{\bbl@starthyphens}}%
3144      \fi
3145      \bbl@exp{\\\bbl@add\\\bbl@starthyphens
3146        {\\\bbl@patterns@lua{\languagename}}}%
3147      % TODO - error/warning if no script
3148      \directlua{
3149        if Babel.script_blocks['\bbl@cl{sbcp}'] then
```

136

```
3150           Babel.loc_to_scr[\the\localeid] =
3151             Babel.script_blocks['\bbl@cl{sbcp}']
3152           Babel.locale_props[\the\localeid].lc = \the\localeid\space
3153           Babel.locale_props[\the\localeid].lg = \the\@nameuse{l@\languagename}\space
3154         end
3155       }%
3156     \fi
3157     \bbl@xin@{ fonts }{ \bbl@KVP@onchar\space}%
3158     \ifin@
3159       \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
3160       \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
3161       \directlua{
3162         if Babel.script_blocks['\bbl@cl{sbcp}'] then
3163           Babel.loc_to_scr[\the\localeid] =
3164             Babel.script_blocks['\bbl@cl{sbcp}']
3165         end}%
3166       \ifx\bbl@mapselect\@undefined  % TODO. almost the same as mapfont
3167         \AtBeginDocument{%
3168           \expandafter\bbl@add\csname selectfont \endcsname{{\bbl@mapselect}}%
3169           {\selectfont}}%
3170         \def\bbl@mapselect{%
3171           \let\bbl@mapselect\relax
3172           \edef\bbl@prefontid{\fontid\font}}%
3173         \def\bbl@mapdir##1{%
3174           {\def\languagename{##1}%
3175            \let\bbl@ifrestoring\@firstoftwo % To avoid font warning
3176            \bbl@switchfont
3177            \directlua{
3178              Babel.locale_props[\the\csname bbl@id@@##1\endcsname]%
3179                     ['/\bbl@prefontid'] = \fontid\font\space}}}%
3180       \fi
3181       \bbl@exp{\\\bbl@add\\\bbl@mapselect{\\\bbl@mapdir{\languagename}}}%
3182     \fi
3183     % TODO - catch non-valid values
3184   \fi
3185   % == mapfont ==
3186   % For bidi texts, to switch the font based on direction
3187   \ifx\bbl@KVP@mapfont\@nil\else
3188     \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}{}%
3189       {\bbl@error{Option `\bbl@KVP@mapfont' unknown for\\%
3190                   mapfont. Use `direction'.%
3191                   {See the manual for details.}}}%
3192     \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
3193     \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
3194     \ifx\bbl@mapselect\@undefined % TODO. See onchar
3195       \AtBeginDocument{%
3196         \expandafter\bbl@add\csname selectfont \endcsname{{\bbl@mapselect}}%
3197         {\selectfont}}%
3198       \def\bbl@mapselect{%
3199         \let\bbl@mapselect\relax
3200         \edef\bbl@prefontid{\fontid\font}}%
3201       \def\bbl@mapdir##1{%
3202         {\def\languagename{##1}%
3203          \let\bbl@ifrestoring\@firstoftwo % avoid font warning
3204          \bbl@switchfont
3205          \directlua{Babel.fontmap
3206            [\the\csname bbl@wdir@##1\endcsname]%
3207            [\bbl@prefontid]=\fontid\font}}}%
3208     \fi
```

137

```
3209     \bbl@exp{\\\bbl@add\\\bbl@mapselect{\\\bbl@mapdir{\languagename}}}%
3210   \fi
3211 % == Line breaking: intraspace, intrapenalty ==
3212 % For CJK, East Asian, Southeast Asian, if interspace in ini
3213 \ifx\bbl@KVP@intraspace\@nil\else % We can override the ini or set
3214   \bbl@csarg\edef{intsp@#2}{\bbl@KVP@intraspace}%
3215 \fi
3216 \bbl@provide@intraspace
3217 %
3218 \bbl@xin@{/e}{/\bbl@cl{lnbrk}}
3219 \ifin@\else\bbl@xin@{/k}{/\bbl@cl{lnbrk}}\fi
3220 \ifin@\bbl@arabicjust\fi
3221 % == Line breaking: hyphenate.other.locale/.script==
3222 \ifx\bbl@lbkflag\@empty
3223   \bbl@ifunset{bbl@hyotl@\languagename}{}%
3224     {\bbl@csarg\bbl@replace{hyotl@\languagename}{ }{,}%
3225     \bbl@startcommands*{\languagename}{}%
3226       \bbl@csarg\bbl@foreach{hyotl@\languagename}{%
3227         \ifcase\bbl@engine
3228           \ifnum##1<257
3229             \SetHyphenMap{\BabelLower{##1}{##1}}%
3230           \fi
3231         \else
3232           \SetHyphenMap{\BabelLower{##1}{##1}}%
3233         \fi}%
3234     \bbl@endcommands}%
3235   \bbl@ifunset{bbl@hyots@\languagename}{}%
3236     {\bbl@csarg\bbl@replace{hyots@\languagename}{ }{,}%
3237     \bbl@csarg\bbl@foreach{hyots@\languagename}{%
3238       \ifcase\bbl@engine
3239         \ifnum##1<257
3240           \global\lccode##1=##1\relax
3241         \fi
3242       \else
3243         \global\lccode##1=##1\relax
3244       \fi}}%
3245 \fi
3246 % == Counters: maparabic ==
3247 % Native digits, if provided in ini (TeX level, xe and lua)
3248 \ifcase\bbl@engine\else
3249   \bbl@ifunset{bbl@dgnat@\languagename}{}%
3250     {\expandafter\ifx\csname bbl@dgnat@\languagename\endcsname\@empty\else
3251       \expandafter\expandafter\expandafter
3252       \bbl@setdigits\csname bbl@dgnat@\languagename\endcsname
3253       \ifx\bbl@KVP@maparabic\@nil\else
3254         \ifx\bbl@latinarabic\@undefined
3255           \expandafter\let\expandafter\@arabic
3256             \csname bbl@counter@\languagename\endcsname
3257         \else    % ie, if layout=counters, which redefines \@arabic
3258           \expandafter\let\expandafter\bbl@latinarabic
3259             \csname bbl@counter@\languagename\endcsname
3260         \fi
3261       \fi
3262     \fi}%
3263 \fi
3264 % == Counters: mapdigits ==
3265 % Native digits (lua level).
3266 \ifodd\bbl@engine
3267   \ifx\bbl@KVP@mapdigits\@nil\else
```

```
3268        \bbl@ifunset{bbl@dgnat@\languagename}{}%
3269          {\RequirePackage{luatexbase}%
3270           \bbl@activate@preotf
3271           \directlua{
3272             Babel = Babel or {}  %%% -> presets in luababel
3273             Babel.digits_mapped = true
3274             Babel.digits = Babel.digits or {}
3275             Babel.digits[\the\localeid] =
3276               table.pack(string.utfvalue('\bbl@cl{dgnat}'))
3277             if not Babel.numbers then
3278               function Babel.numbers(head)
3279                 local LOCALE = luatexbase.registernumber'bbl@attr@locale'
3280                 local GLYPH = node.id'glyph'
3281                 local inmath = false
3282                 for item in node.traverse(head) do
3283                   if not inmath and item.id == GLYPH then
3284                     local temp = node.get_attribute(item, LOCALE)
3285                     if Babel.digits[temp] then
3286                       local chr = item.char
3287                       if chr > 47 and chr < 58 then
3288                         item.char = Babel.digits[temp][chr-47]
3289                       end
3290                     end
3291                   elseif item.id == node.id'math' then
3292                     inmath = (item.subtype == 0)
3293                   end
3294                 end
3295                 return head
3296               end
3297             end
3298          }}%
3299      \fi
3300    \fi
3301    % == Counters: alph, Alph ==
3302    % What if extras<lang> contains a \babel@save\@alph? It won't be
3303    % restored correctly when exiting the language, so we ignore
3304    % this change with the \bbl@alph@saved trick.
3305    \ifx\bbl@KVP@alph\@nil\else
3306      \toks@\expandafter\expandafter\expandafter{%
3307        \csname extras\languagename\endcsname}%
3308      \bbl@exp{%
3309        \def\<extras\languagename>{%
3310          \let\\\bbl@alph@saved\\\@alph
3311          \the\toks@
3312          \let\\\@alph\\\bbl@alph@saved
3313          \\\babel@save\\\@alph
3314          \let\\\@alph\<bbl@cntr@\bbl@KVP@alph @\languagename>}}%
3315    \fi
3316    \ifx\bbl@KVP@Alph\@nil\else
3317      \toks@\expandafter\expandafter\expandafter{%
3318        \csname extras\languagename\endcsname}%
3319      \bbl@exp{%
3320        \def\<extras\languagename>{%
3321          \let\\\bbl@Alph@saved\\\@Alph
3322          \the\toks@
3323          \let\\\@Alph\\\bbl@Alph@saved
3324          \\\babel@save\\\@Alph
3325          \let\\\@Alph\<bbl@cntr@\bbl@KVP@Alph @\languagename>}}%
3326    \fi
```

```
3327  % == require.babel in ini ==
3328  % To load or reaload the babel-*.tex, if require.babel in ini
3329  \ifx\bbl@beforestart\relax\else  % But not in doc aux or body
3330    \bbl@ifunset{bbl@rqtex@\languagename}{}%
3331      {\expandafter\ifx\csname bbl@rqtex@\languagename\endcsname\@empty\else
3332        \let\BabelBeforeIni\@gobbletwo
3333        \chardef\atcatcode=\catcode`\@
3334        \catcode`\@=11\relax
3335        \bbl@input@texini{\bbl@cs{rqtex@\languagename}}%
3336        \catcode`\@=\atcatcode
3337        \let\atcatcode\relax
3338      \fi}%
3339  \fi
3340  % == Release saved transforms ==
3341  \bbl@release@transforms\relax % \relax closes the last item.
3342  % == main ==
3343  \ifx\bbl@KVP@main\@nil  % Restore only if not 'main'
3344    \let\languagename\bbl@savelangname
3345    \chardef\localeid\bbl@savelocaleid\relax
3346  \fi}
```

Depending on whether or not the language exists, we define two macros.

```
3347  \def\bbl@provide@new#1{%
3348    \@namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
3349    \@namedef{extras#1}{}%
3350    \@namedef{noextras#1}{}%
3351    \bbl@startcommands*{#1}{captions}%
3352      \ifx\bbl@KVP@captions\@nil %       and also if import, implicit
3353        \def\bbl@tempb##1{%               elt for \bbl@captionslist
3354          \ifx##1\@empty\else
3355            \bbl@exp{%
3356              \\\SetString\\##1{%
3357                \\\bbl@nocaption{\bbl@stripslash##1}{#1\bbl@stripslash##1}}}%
3358            \expandafter\bbl@tempb
3359          \fi}%
3360        \expandafter\bbl@tempb\bbl@captionslist\@empty
3361      \else
3362        \ifx\bbl@initoload\relax
3363          \bbl@read@ini{\bbl@KVP@captions}2%  % Here letters cat = 11
3364        \else
3365          \bbl@read@ini{\bbl@initoload}2%     % Same
3366        \fi
3367      \fi
3368    \StartBabelCommands*{#1}{date}%
3369      \ifx\bbl@KVP@import\@nil
3370        \bbl@exp{%
3371          \\\SetString\\\today{\\\bbl@nocaption{today}{#1today}}}%
3372      \else
3373        \bbl@savetoday
3374        \bbl@savedate
3375      \fi
3376    \bbl@endcommands
3377    \bbl@load@basic{#1}%
3378    % == hyphenmins == (only if new)
3379    \bbl@exp{%
3380      \gdef\<#1hyphenmins>{%
3381        {\bbl@ifunset{bbl@lfthm@#1}{2}{\bbl@cs{lfthm@#1}}}%
3382        {\bbl@ifunset{bbl@rgthm@#1}{3}{\bbl@cs{rgthm@#1}}}}}%
3383    % == hyphenrules ==
```

```
3384   \bbl@provide@hyphens{#1}%
3385   % == frenchspacing == (only if new)
3386   \bbl@ifunset{bbl@frspc@#1}{}%
3387     {\edef\bbl@tempa{\bbl@cl{frspc}}%
3388      \edef\bbl@tempa{\expandafter\@car\bbl@tempa\@nil}%
3389      \if u\bbl@tempa          % do nothing
3390      \else\if n\bbl@tempa     % non french
3391        \expandafter\bbl@add\csname extras#1\endcsname{%
3392          \let\bbl@elt\bbl@fs@elt@i
3393          \bbl@fs@chars}%
3394      \else\if y\bbl@tempa       % french
3395        \expandafter\bbl@add\csname extras#1\endcsname{%
3396          \let\bbl@elt\bbl@fs@elt@ii
3397          \bbl@fs@chars}%
3398      \fi\fi\fi}%
3399   %
3400   \ifx\bbl@KVP@main\@nil\else
3401     \expandafter\main@language\expandafter{#1}%
3402   \fi}
3403 % A couple of macros used above, to avoid hashes #######...
3404 \def\bbl@fs@elt@i#1#2#3{%
3405   \ifnum\sfcode`#1=#2\relax
3406     \babel@savevariable{\sfcode`#1}%
3407     \sfcode`#1=#3\relax
3408   \fi}%
3409 \def\bbl@fs@elt@ii#1#2#3{%
3410   \ifnum\sfcode`#1=#3\relax
3411     \babel@savevariable{\sfcode`#1}%
3412     \sfcode`#1=#2\relax
3413   \fi}%
3414 %
3415 \def\bbl@provide@renew#1{%
3416   \ifx\bbl@KVP@captions\@nil\else
3417     \StartBabelCommands*{#1}{captions}%
3418       \bbl@read@ini{\bbl@KVP@captions}2%   % Here all letters cat = 11
3419     \EndBabelCommands
3420   \fi
3421   \ifx\bbl@KVP@import\@nil\else
3422     \StartBabelCommands*{#1}{date}%
3423       \bbl@savetoday
3424       \bbl@savedate
3425     \EndBabelCommands
3426   \fi
3427   % == hyphenrules ==
3428   \ifx\bbl@lbkflag\@empty
3429     \bbl@provide@hyphens{#1}%
3430   \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.)

```
3431 \def\bbl@load@basic#1{%
3432   \bbl@ifunset{bbl@inidata@\languagename}{}%
3433     {\getlocaleproperty\bbl@tempa{\languagename}{identification/load.level}%
3434      \ifcase\bbl@tempa
3435        \bbl@csarg\let{lname@\languagename}\relax
3436      \fi}%
3437   \bbl@ifunset{bbl@lname@#1}%
3438     {\def\BabelBeforeIni##1##2{%
```

141

```
3439        \begingroup
3440          \let\bbl@ini@captions@aux\@gobbletwo
3441          \def\bbl@inidate ####1.####2.####3.####4\relax ####5####6{}%
3442          \bbl@read@ini{##1}1%
3443          \ifx\bbl@initoload\relax\endinput\fi
3444        \endgroup}%
3445      \begingroup         % boxed, to avoid extra spaces:
3446        \ifx\bbl@initoload\relax
3447          \bbl@input@texini{#1}%
3448        \else
3449          \setbox\z@\hbox{\BabelBeforeIni{\bbl@initoload}{}}%
3450        \fi
3451      \endgroup}%
3452     {}}}
```

The hyphenrules option is handled with an auxiliary macro.

```
3453 \def\bbl@provide@hyphens#1{%
3454   \let\bbl@tempa\relax
3455   \ifx\bbl@KVP@hyphenrules\@nil\else
3456     \bbl@replace\bbl@KVP@hyphenrules{ }{,}%
3457     \bbl@foreach\bbl@KVP@hyphenrules{%
3458       \ifx\bbl@tempa\relax    % if not yet found
3459         \bbl@ifsamestring{##1}{+}%
3460           {{\bbl@exp{\\\addlanguage\<l@##1>}}}%
3461           {}%
3462         \bbl@ifunset{l@##1}%
3463           {}%
3464           {\bbl@exp{\let\bbl@tempa\<l@##1>}}%
3465       \fi}%
3466   \fi
3467   \ifx\bbl@tempa\relax %          if no opt or no language in opt found
3468     \ifx\bbl@KVP@import\@nil
3469       \ifx\bbl@initoload\relax\else
3470         \bbl@exp{%               and hyphenrules is not empty
3471           \\\bbl@ifblank{\bbl@cs{hyphr@#1}}%
3472             {}%
3473             {\let\\\bbl@tempa\<l@\bbl@cl{hyphr}>}}%
3474       \fi
3475     \else % if importing
3476       \bbl@exp{%                  and hyphenrules is not empty
3477         \\\bbl@ifblank{\bbl@cs{hyphr@#1}}%
3478           {}%
3479           {\let\\\bbl@tempa\<l@\bbl@cl{hyphr}>}}%
3480     \fi
3481   \fi
3482   \bbl@ifunset{bbl@tempa}%        ie, relax or undefined
3483     {\bbl@ifunset{l@#1}%          no hyphenrules found - fallback
3484       {\bbl@exp{\\\adddialect\<l@#1>\language}}%
3485       {}}%                        so, l@<lang> is ok - nothing to do
3486     {\bbl@exp{\\\adddialect\<l@#1>\bbl@tempa}}}% found in opt list or ini
```

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

```
3487 \def\bbl@input@texini#1{%
3488   \bbl@bsphack
3489     \bbl@exp{%
3490       \catcode`\\\%=14 \catcode`\\\\=0
3491       \catcode`\\\{=1  \catcode`\\\}=2
3492       \lowercase{\\\InputIfFileExists{babel-#1.tex}{}{}}%
3493       \catcode`\\\%=\the\catcode`\%\relax
```

```
3494        \catcode`\\\\=\the\catcode`\\\relax
3495        \catcode`\\\{=\the\catcode`\{\relax
3496        \catcode`\\\}=\the\catcode`\}\relax}%
3497    \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.

```
3498 \def\bbl@iniline#1\bbl@iniline{%
3499   \@ifnextchar[\bbl@inisect{\@ifnextchar;\bbl@iniskip\bbl@inistore}#1\@@}% ]
3500 \def\bbl@inisect[#1]#2\@@{\def\bbl@section{#1}}%
3501 \def\bbl@iniskip#1\@@{}%       if starts with ;
3502 \def\bbl@inistore#1=#2\@@{%       full (default)
3503   \bbl@trim@def\bbl@tempa{#1}%
3504   \bbl@trim\toks@{#2}%
3505   \bbl@ifunset{bbl@KVP@\bbl@section/\bbl@tempa}%
3506     {\bbl@exp{%
3507       \\\g@addto@macro\\\bbl@inidata{%
3508         \\\bbl@elt{\bbl@section}{\bbl@tempa}{\the\toks@}}}}%
3509     {}}%
3510 \def\bbl@inistore@min#1=#2\@@{%  minimal (maybe set in \bbl@read@ini)
3511   \bbl@trim@def\bbl@tempa{#1}%
3512   \bbl@trim\toks@{#2}%
3513   \bbl@xin@{.identification.}{.\bbl@section.}%
3514   \ifin@
3515     \bbl@exp{\\\g@addto@macro\\\bbl@inidata{%
3516       \\\bbl@elt{identification}{\bbl@tempa}{\the\toks@}}}%
3517   \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.

```
3518 \ifx\bbl@readstream\@undefined
3519   \csname newread\endcsname\bbl@readstream
3520 \fi
3521 \def\bbl@read@ini#1#2{%
3522   \openin\bbl@readstream=babel-#1.ini
3523   \ifeof\bbl@readstream
3524     \bbl@error
3525       {There is no ini file for the requested language\\%
3526        (#1). Perhaps you misspelled it or your installation\\%
3527        is not complete.}%
3528       {Fix the name or reinstall babel.}%
3529   \else
3530     % Store ini data in \bbl@inidata
3531     \catcode`\[=12 \catcode`\]=12 \catcode`\==12 \catcode`\&=12
3532     \catcode`\;=12 \catcode`\|=12 \catcode`\%=14 \catcode`\-=12
3533     \bbl@info{Importing
3534                 \ifcase#2font and identification \or basic \fi
3535                  data for \languagename\\%
3536               from babel-#1.ini. Reported}%
3537     \ifnum#2=\z@
3538       \global\let\bbl@inidata\@empty
3539       \let\bbl@inistore\bbl@inistore@min     % Remember it's local
3540     \fi
3541     \def\bbl@section{identification}%
```

143

```
3542      \bbl@exp{\\\bbl@inistore tag.ini=#1\\\@@}%
3543      \bbl@inistore load.level=#2\@@
3544      \loop
3545      \if T\ifeof\bbl@readstream F\fi T\relax % Trick, because inside \loop
3546        \endlinechar\m@ne
3547        \read\bbl@readstream to \bbl@line
3548        \endlinechar`\^^M
3549        \ifx\bbl@line\@empty\else
3550          \expandafter\bbl@iniline\bbl@line\bbl@iniline
3551        \fi
3552      \repeat
3553      % Process stored data
3554      \bbl@csarg\xdef{lini@\languagename}{#1}%
3555      \let\bbl@savestrings\@empty
3556      \let\bbl@savetoday\@empty
3557      \let\bbl@savedate\@empty
3558      \def\bbl@elt##1##2##3{%
3559        \def\bbl@section{##1}%
3560        \in@{=date.}{=##1}% Find a better place
3561        \ifin@
3562          \bbl@ini@calendar{##1}%
3563        \fi
3564        \global\bbl@csarg\let{bbl@KVP@##1/##2}\relax
3565        \bbl@ifunset{bbl@inikv@##1}{}%
3566          {\csname bbl@inikv@##1\endcsname{##2}{##3}}}%
3567      \bbl@inidata
3568      % 'Export' data
3569      \bbl@ini@exports{#2}%
3570      \global\bbl@csarg\let{inidata@\languagename}\bbl@inidata
3571      \global\let\bbl@inidata\@empty
3572      \bbl@exp{\\\bbl@add@list\\\bbl@ini@loaded{\languagename}}%
3573      \bbl@toglobal\bbl@ini@loaded
3574    \fi}
```

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

```
3575 \def\bbl@ini@calendar#1{%
3576 \lowercase{\def\bbl@tempa{=#1=}}%
3577 \bbl@replace\bbl@tempa{=date.gregorian}{}%
3578 \bbl@replace\bbl@tempa{=date.}{}%
3579 \in@{.licr=}{#1=}%
3580 \ifin@
3581   \ifcase\bbl@engine
3582     \bbl@replace\bbl@tempa{.licr=}{}%
3583   \else
3584     \let\bbl@tempa\relax
3585   \fi
3586 \fi
3587 \ifx\bbl@tempa\relax\else
3588   \bbl@replace\bbl@tempa{=}{}%
3589   \bbl@exp{%
3590     \def\<bbl@inikv@#1>####1####2{%
3591       \\\bbl@inidate####1...\relax{####2}{\bbl@tempa}}}%
3592 \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).

```
3593 \def\bbl@renewinikey#1/#2\@@#3{%
```

```
3594    \edef\bbl@tempa{\zap@space #1 \@empty}%    section
3595    \edef\bbl@tempb{\zap@space #2 \@empty}%    key
3596    \bbl@trim\toks@{#3}%                        value
3597    \bbl@exp{%
3598      \global\let\<bbl@KVP@\bbl@tempa/\bbl@tempb>\\\@empty % just a flag
3599      \\\g@addto@macro\\\bbl@inidata{%
3600        \\\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.

```
3601 \def\bbl@exportkey#1#2#3{%
3602    \bbl@ifunset{bbl@@kv@#2}%
3603      {\bbl@csarg\gdef{#1@\languagename}{#3}}%
3604      {\expandafter\ifx\csname bbl@@kv@#2\endcsname\@empty
3605        \bbl@csarg\gdef{#1@\languagename}{#3}%
3606      \else
3607        \bbl@exp{\global\let\<bbl@#1@\languagename>\<bbl@@kv@#2>}%
3608      \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.

```
3609 \def\bbl@iniwarning#1{%
3610    \bbl@ifunset{bbl@@kv@identification.warning#1}{}%
3611      {\bbl@warning{%
3612        From babel-\bbl@cs{lini@\languagename}.ini:\\%
3613        \bbl@cs{@kv@identification.warning#1}\\%
3614        Reported }}}
3615 %
3616 \let\bbl@release@transforms\@empty
3617 %
3618 \def\bbl@ini@exports#1{%
3619   % Identification always exported
3620   \bbl@iniwarning{}%
3621   \ifcase\bbl@engine
3622     \bbl@iniwarning{.pdflatex}%
3623   \or
3624     \bbl@iniwarning{.lualatex}%
3625   \or
3626     \bbl@iniwarning{.xelatex}%
3627   \fi%
3628   \bbl@exportkey{elname}{identification.name.english}{}%
3629   \bbl@exp{\\\bbl@exportkey{lname}{identification.name.opentype}%
3630     {\csname bbl@elname@\languagename\endcsname}}%
3631   \bbl@exportkey{tbcp}{identification.tag.bcp47}{}%
3632   \bbl@exportkey{lbcp}{identification.language.tag.bcp47}{}%
3633   \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
3634   \bbl@exportkey{esname}{identification.script.name}{}%
3635   \bbl@exp{\\\bbl@exportkey{sname}{identification.script.name.opentype}%
3636     {\csname bbl@esname@\languagename\endcsname}}%
3637   \bbl@exportkey{sbcp}{identification.script.tag.bcp47}{}%
3638   \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
3639   % Also maps bcp47 -> languagename
3640   \ifbbl@bcptoname
3641     \bbl@csarg\xdef{bcp@map@\bbl@cl{tbcp}}{\languagename}%
3642   \fi
3643   % Conditional
3644   \ifnum#1>\z@          % 0 = only info, 1, 2 = basic, (re)new
3645     \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
```

```
3646      \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
3647      \bbl@exportkey{lfthm}{typography.lefthyphenmin}{2}%
3648      \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
3649      \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
3650      \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
3651      \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
3652      \bbl@exportkey{intsp}{typography.intraspace}{}%
3653      \bbl@exportkey{chrng}{characters.ranges}{}%
3654      \bbl@exportkey{dgnat}{numbers.digits.native}{}%
3655      \bbl@exportkey{frspc}{typography.frenchspacing}{u}%
3656      \ifnum#1=\tw@          % only (re)new
3657        \bbl@exportkey{rqtex}{identification.require.babel}{}%
3658        \bbl@toglobal\bbl@savetoday
3659        \bbl@toglobal\bbl@savedate
3660        \bbl@savestrings
3661      \fi
3662    \fi}
```

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

```
3663 \def\bbl@inikv#1#2{%      key=value
3664   \toks@{#2}%              This hides #'s from ini values
3665   \bbl@csarg\edef{@kv@\bbl@section.#1}{\the\toks@}}
```

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

```
3666 \let\bbl@inikv@identification\bbl@inikv
3667 \let\bbl@inikv@typography\bbl@inikv
3668 \let\bbl@inikv@characters\bbl@inikv
3669 \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'.

```
3670 \def\bbl@inikv@counters#1#2{%
3671   \bbl@ifsamestring{#1}{digits}%
3672     {\bbl@error{The counter name 'digits' is reserved for mapping\\%
3673                 decimal digits}%
3674                {Use another name.}}%
3675     {}%
3676   \def\bbl@tempc{#1}%
3677   \bbl@trim@def{\bbl@tempb*}{#2}%
3678   \in@{.1$}{#1$}%
3679   \ifin@
3680     \bbl@replace\bbl@tempc{.1}{}%
3681     \bbl@csarg\protected@xdef{cntr@\bbl@tempc @\languagename}{%
3682       \noexpand\bbl@alphnumeral{\bbl@tempc}}%
3683   \fi
3684   \in@{.F.}{#1}%
3685   \ifin@\else\in@{.S.}{#1}\fi
3686   \ifin@
3687     \bbl@csarg\protected@xdef{cntr@#1@\languagename}{\bbl@tempb*}%
3688   \else
3689     \toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa
3690     \expandafter\bbl@buildifcase\bbl@tempb* \\ % Space after \\
3691     \bbl@csarg{\global\expandafter\let}{cntr@#1@\languagename}\bbl@tempa
3692   \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.

```
3693 \ifcase\bbl@engine
3694   \bbl@csarg\def{inikv@captions.licr}#1#2{%
3695     \bbl@ini@captions@aux{#1}{#2}}
3696 \else
3697   \def\bbl@inikv@captions#1#2{%
3698     \bbl@ini@captions@aux{#1}{#2}}
3699 \fi
```

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

```
3700 \def\bbl@ini@captions@template#1#2{% string language tempa=capt-name
3701   \bbl@replace\bbl@tempa{.template}{}%
3702   \def\bbl@toreplace{#1{}}%
3703   \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
3704   \bbl@replace\bbl@toreplace{[[}{\csname}%
3705   \bbl@replace\bbl@toreplace{[}{\csname the}%
3706   \bbl@replace\bbl@toreplace{]]}{name\endcsname{}}%
3707   \bbl@replace\bbl@toreplace{]}{\endcsname{}}%
3708   \bbl@xin@{,\bbl@tempa,}{,chapter,appendix,part,}%
3709   \ifin@
3710     \@nameuse{bbl@patch\bbl@tempa}%
3711     \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
3712   \fi
3713   \bbl@xin@{,\bbl@tempa,}{,figure,table,}%
3714   \ifin@
3715     \toks@\expandafter{\bbl@toreplace}%
3716     \bbl@exp{\gdef\<fnum@\bbl@tempa>{\the\toks@}}%
3717   \fi}
3718 \def\bbl@ini@captions@aux#1#2{%
3719   \bbl@trim@def\bbl@tempa{#1}%
3720   \bbl@xin@{.template}{\bbl@tempa}%
3721   \ifin@
3722     \bbl@ini@captions@template{#2}\languagename
3723   \else
3724     \bbl@ifblank{#2}%
3725       {\bbl@exp{%
3726         \toks@{\\\bbl@nocaption{\bbl@tempa}{\languagename\bbl@tempa name}}}}%
3727       {\bbl@trim\toks@{#2}}%
3728     \bbl@exp{%
3729       \\\bbl@add\\\bbl@savestrings{%
3730         \\\SetString\<\bbl@tempa name>{\the\toks@}}}%
3731     \toks@\expandafter{\bbl@captionslist}%
3732     \bbl@exp{\\\in@{\<\bbl@tempa name>}{\the\toks@}}%
3733     \ifin@\else
3734       \bbl@exp{%
3735         \\\bbl@add\<bbl@extracaps@\languagename>{\<\bbl@tempa name>}%
3736         \\\bbl@toglobal\<bbl@extracaps@\languagename>}%
3737     \fi
3738   \fi}
```

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

```
3739 \def\bbl@list@the{%
3740   part,chapter,section,subsection,subsubsection,paragraph,%
3741   subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,%
3742   table,page,footnote,mpfootnote,mpfn}
3743 \def\bbl@map@cnt#1{%  #1:roman,etc, // #2:enumi,etc
3744   \bbl@ifunset{bbl@map@#1@\languagename}%
3745     {\@nameuse{#1}}%
3746     {\@nameuse{bbl@map@#1@\languagename}}}
3747 \def\bbl@inikv@labels#1#2{%
```

```
3748   \in@{.map}{#1}%
3749   \ifin@
3750     \ifx\bbl@KVP@labels\@nil\else
3751       \bbl@xin@{ map }{ \bbl@KVP@labels\space}%
3752       \ifin@
3753         \def\bbl@tempc{#1}%
3754         \bbl@replace\bbl@tempc{.map}{}%
3755         \in@{,#2,}{,arabic,roman,Roman,alph,Alph,fnsymbol,}%
3756         \bbl@exp{%
3757           \gdef\<bbl@map@\bbl@tempc @\languagename>%
3758             {\ifin@\<#2>\else\\\localcounter{#2}\fi}}%
3759         \bbl@foreach\bbl@list@the{%
3760           \bbl@ifunset{the##1}{}%
3761             {\bbl@exp{\let\\\bbl@tempd\<the##1>}%
3762              \bbl@exp{%
3763                \\\bbl@sreplace\<the##1>%
3764                  {\<bbl@tempc>{##1}}{\\\bbl@map@cnt{\bbl@tempc}{##1}}%
3765                \\\bbl@sreplace\<the##1>%
3766                  {\<\@empty @\bbl@tempc>\<c@##1>}{\\\bbl@map@cnt{\bbl@tempc}{##1}}}%
3767              \expandafter\ifx\csname the##1\endcsname\bbl@tempd\else
3768                \toks@\expandafter\expandafter\expandafter{%
3769                  \csname the##1\endcsname}%
3770                \expandafter\xdef\csname the##1\endcsname{{\the\toks@}}%
3771              \fi}}%
3772       \fi
3773     \fi
3774   %
3775   \else
3776     %
3777     % The following code is still under study. You can test it and make
3778     % suggestions. Eg, enumerate.2 = ([enumi]).([enumii]). It's
3779     % language dependent.
3780     \in@{enumerate.}{#1}%
3781     \ifin@
3782       \def\bbl@tempa{#1}%
3783       \bbl@replace\bbl@tempa{enumerate.}{}%
3784       \def\bbl@toreplace{#2}%
3785       \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
3786       \bbl@replace\bbl@toreplace{[}{\csname the}%
3787       \bbl@replace\bbl@toreplace{]}{\endcsname{}}%
3788       \toks@\expandafter{\bbl@toreplace}%
3789       \bbl@exp{%
3790         \\\bbl@add\<extras\languagename>{%
3791           \\\babel@save\<labelenum\romannumeral\bbl@tempa>%
3792           \def\<labelenum\romannumeral\bbl@tempa>{\the\toks@}}%
3793         \\\bbl@toglobal\<extras\languagename>}%
3794     \fi
3795   \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.

```
3796 \def\bbl@chaptype{chapter}
3797 \ifx\@makechapterhead\@undefined
3798   \let\bbl@patchchapter\relax
3799 \else\ifx\thechapter\@undefined
3800   \let\bbl@patchchapter\relax
3801 \else\ifx\ps@headings\@undefined
```

```
3802    \let\bbl@patchchapter\relax
3803 \else
3804    \def\bbl@patchchapter{%
3805        \global\let\bbl@patchchapter\relax
3806        \bbl@add\appendix{\def\bbl@chaptype{appendix}}% Not harmful, I hope
3807        \bbl@toglobal\appendix
3808        \bbl@sreplace\ps@headings
3809            {\@chapapp\ \thechapter}%
3810            {\bbl@chapterformat}%
3811        \bbl@toglobal\ps@headings
3812        \bbl@sreplace\chaptermark
3813            {\@chapapp\ \thechapter}%
3814            {\bbl@chapterformat}%
3815        \bbl@toglobal\chaptermark
3816        \bbl@sreplace\@makechapterhead
3817            {\@chapapp\space\thechapter}%
3818            {\bbl@chapterformat}%
3819        \bbl@toglobal\@makechapterhead
3820        \gdef\bbl@chapterformat{%
3821            \bbl@ifunset{bbl@\bbl@chaptype fmt@\languagename}%
3822                {\@chapapp\space\thechapter}
3823                {\@nameuse{bbl@\bbl@chaptype fmt@\languagename}}}}
3824    \let\bbl@patchappendix\bbl@patchchapter
3825 \fi\fi\fi
3826 \ifx\@part\@undefined
3827    \let\bbl@patchpart\relax
3828 \else
3829    \def\bbl@patchpart{%
3830        \global\let\bbl@patchpart\relax
3831        \bbl@sreplace\@part
3832            {\partname\nobreakspace\thepart}%
3833            {\bbl@partformat}%
3834        \bbl@toglobal\@part
3835        \gdef\bbl@partformat{%
3836            \bbl@ifunset{bbl@partfmt@\languagename}%
3837                {\partname\nobreakspace\thepart}
3838                {\@nameuse{bbl@partfmt@\languagename}}}}
3839 \fi
```

Date. TODO. Document

```
3840 % Arguments are _not_ protected.
3841 \let\bbl@calendar\@empty
3842 \DeclareRobustCommand\localedate[1][]{\bbl@localedate{#1}}
3843 \def\bbl@localedate#1#2#3#4{%
3844    \begingroup
3845    \ifx\@empty#1\@empty\else
3846        \let\bbl@ld@calendar\@empty
3847        \let\bbl@ld@variant\@empty
3848        \edef\bbl@tempa{\zap@space#1 \@empty}%
3849        \def\bbl@tempb##1=##2\@@{\@namedef{bbl@ld@##1}{##2}}%
3850        \bbl@foreach\bbl@tempa{\bbl@tempb##1\@@}%
3851        \edef\bbl@calendar{%
3852            \bbl@ld@calendar
3853            \ifx\bbl@ld@variant\@empty\else
3854                .\bbl@ld@variant
3855            \fi}%
3856        \bbl@replace\bbl@calendar{gregorian}{}%
3857    \fi
3858    \bbl@cased
```

```
3859        {\@nameuse{bbl@date@\languagename @\bbl@calendar}{#2}{#3}{#4}}%
3860    \endgroup}
3861 % eg: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
3862 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
3863    \bbl@trim@def\bbl@tempa{#1.#2}%
3864    \bbl@ifsamestring{\bbl@tempa}{months.wide}%       to savedate
3865      {\bbl@trim@def\bbl@tempa{#3}%
3866       \bbl@trim\toks@{#5}%
3867       \@temptokena\expandafter{\bbl@savedate}%
3868       \bbl@exp{%    Reverse order - in ini last wins
3869         \def\\\bbl@savedate{%
3870           \\\SetString\<month\romannumeral\bbl@tempa#6name>{\the\toks@}%
3871           \the\@temptokena}}%
3872      {\bbl@ifsamestring{\bbl@tempa}{date.long}%       defined now
3873        {\lowercase{\def\bbl@tempb{#6}}%
3874         \bbl@trim@def\bbl@toreplace{#5}%
3875         \bbl@TG@@date
3876         \bbl@ifunset{bbl@date@\languagename @}%
3877           {\global\bbl@csarg\let{date@\languagename @}\bbl@toreplace
3878           % TODO. Move to a better place.
3879            \bbl@exp{%
3880              \gdef\<\languagename date>{\\\protect\<\languagename date >}%
3881              \gdef\<\languagename date >####1####2####3{%
3882                \\\bbl@usedategrouptrue
3883                \<bbl@ensure@\languagename>{%
3884                  \\\localedate{####1}{####2}{####3}}}%
3885              \\\bbl@add\\\bbl@savetoday{%
3886                \\\SetString\\\today{%
3887                  \<\languagename date>%
3888                    {\\\the\year}{\\\the\month}{\\\the\day}}}}}%
3889           {}%
3890         \ifx\bbl@tempb\@empty\else
3891           \global\bbl@csarg\let{date@\languagename @}\bbl@tempb}\bbl@toreplace
3892         \fi}%
3893        {}}}
```

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.

```
3894 \let\bbl@calendar\@empty
3895 \newcommand\BabelDateSpace{\nobreakspace}
3896 \newcommand\BabelDateDot{.\@}  % TODO. \let instead of repeating
3897 \newcommand\BabelDated[1]{{\number#1}}
3898 \newcommand\BabelDatedd[1]{{\ifnum#1<10 0\fi\number#1}}
3899 \newcommand\BabelDateM[1]{{\number#1}}
3900 \newcommand\BabelDateMM[1]{{\ifnum#1<10 0\fi\number#1}}
3901 \newcommand\BabelDateMMMM[1]{{%
3902    \csname month\romannumeral#1\bbl@calendar name\endcsname}}%
3903 \newcommand\BabelDatey[1]{{\number#1}}%
3904 \newcommand\BabelDateyy[1]{{%
3905    \ifnum#1<10 0\number#1 %
3906    \else\ifnum#1<100 \number#1 %
3907    \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3908    \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
3909    \else
3910      \bbl@error
3911        {Currently two-digit years are restricted to the\\
3912         range 0-9999.}%
3913        {There is little you can do. Sorry.}%
```

```
3914    \fi\fi\fi\fi}}
3915 \newcommand\BabelDateyyyy[1]{{\number#1}} % FIXME - add leading 0
3916 \def\bbl@replace@finish@iii#1{%
3917    \bbl@exp{\def\\#1####1####2####3{\the\toks@}}}
3918 \def\bbl@TG@@date{%
3919    \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace{}}%
3920    \bbl@replace\bbl@toreplace{[.]}{\BabelDateDot{}}%
3921    \bbl@replace\bbl@toreplace{[d]}{\BabelDated{####3}}%
3922    \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{####3}}%
3923    \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{####2}}%
3924    \bbl@replace\bbl@toreplace{[MM]}{\BabelDateMM{####2}}%
3925    \bbl@replace\bbl@toreplace{[MMMM]}{\BabelDateMMMM{####2}}%
3926    \bbl@replace\bbl@toreplace{[y]}{\BabelDatey{####1}}%
3927    \bbl@replace\bbl@toreplace{[yy]}{\BabelDateyy{####1}}%
3928    \bbl@replace\bbl@toreplace{[yyyy]}{\BabelDateyyyy{####1}}%
3929    \bbl@replace\bbl@toreplace{[y|}{\bbl@datecntr[####1|}%
3930    \bbl@replace\bbl@toreplace{[m|}{\bbl@datecntr[####2|}%
3931    \bbl@replace\bbl@toreplace{[d|}{\bbl@datecntr[####3|}%
3932 % Note after \bbl@replace \toks@ contains the resulting string.
3933 % TODO - Using this implicit behavior doesn't seem a good idea.
3934    \bbl@replace@finish@iii\bbl@toreplace}
3935 \def\bbl@datecntr{\expandafter\bbl@xdatecntr\expandafter}
3936 \def\bbl@xdatecntr[#1|#2]{\localenumeral{#2}{#1}}
```

Transforms.

```
3937 \let\bbl@release@transforms\@empty
3938 \@namedef{bbl@inikv@transforms.prehyphenation}{%
3939    \bbl@transforms\babelprehyphenation}
3940 \@namedef{bbl@inikv@transforms.posthyphenation}{%
3941    \bbl@transforms\babelposthyphenation}
3942 \def\bbl@transforms@aux#1#2#3,#4\relax{#1{#2}{#3}{#4}}
3943 \begingroup
3944    \catcode`\%=12
3945    \catcode`\&=14
3946    \gdef\bbl@transforms#1#2#3{&%
3947       \ifx\bbl@KVP@transforms\@nil\else
3948          \directlua{
3949             str = [==[#2]==]
3950             str = str:gsub('%.%d+%.%d+$', '')
3951             tex.print([[\def\string\babeltempa{]] .. str .. [[}]])
3952          }&%
3953          \bbl@xin@{,\babeltempa,}{,\bbl@KVP@transforms,}&%
3954          \ifin@
3955             \in@{.0$}{#2$}&%
3956             \ifin@
3957                \g@addto@macro\bbl@release@transforms{&%
3958                   \relax\bbl@transforms@aux#1{\languagename}{#3}}&%
3959             \else
3960                \g@addto@macro\bbl@release@transforms{, {#3}}&%
3961             \fi
3962          \fi
3963       \fi}
3964 \endgroup
```

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

```
3965 \def\bbl@provide@lsys#1{%
3966    \bbl@ifunset{bbl@lname@#1}%
3967       {\bbl@load@info{#1}}%
```

```
3968      {}%
3969   \bbl@csarg\let{lsys@#1}\@empty
3970   \bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}{}%
3971   \bbl@ifunset{bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{}%
3972   \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
3973   \bbl@ifunset{bbl@lname@#1}{}%
3974     {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
3975   \ifcase\bbl@engine\or\or
3976     \bbl@ifunset{bbl@prehc@#1}{}%
3977       {\bbl@exp{\\\bbl@ifblank{\bbl@cs{prehc@#1}}}%
3978         {}%
3979         {\ifx\bbl@xenohyph\@undefined
3980            \let\bbl@xenohyph\bbl@xenohyph@d
3981            \ifx\AtBeginDocument\@notprerr
3982              \expandafter\@secondoftwo  % to execute right now
3983            \fi
3984            \AtBeginDocument{%
3985              \expandafter\bbl@add
3986              \csname selectfont \endcsname{\bbl@xenohyph}%
3987              \expandafter\selectlanguage\expandafter{\languagename}%
3988              \expandafter\bbl@toglobal\csname selectfont \endcsname}%
3989         \fi}}%
3990   \fi
3991   \bbl@csarg\bbl@toglobal{lsys@#1}}
3992 \def\bbl@xenohyph@d{%
3993   \bbl@ifset{bbl@prehc@\languagename}%
3994     {\ifnum\hyphenchar\font=\defaulthyphenchar
3995        \iffontchar\font\bbl@cl{prehc}\relax
3996          \hyphenchar\font\bbl@cl{prehc}\relax
3997        \else\iffontchar\font"200B
3998          \hyphenchar\font"200B
3999        \else
4000          \bbl@warning
4001            {Neither 0 nor ZERO WIDTH SPACE are available\\%
4002             in the current font, and therefore the hyphen\\%
4003             will be printed. Try changing the fontspec's\\%
4004             'HyphenChar' to another value, but be aware\\%
4005             this setting is not safe (see the manual)}%
4006          \hyphenchar\font\defaulthyphenchar
4007        \fi\fi
4008     \fi}%
4009     {\hyphenchar\font\defaulthyphenchar}}
4010   % \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).

```
4011 \def\bbl@load@info#1{%
4012   \def\BabelBeforeIni##1##2{%
4013     \begingroup
4014       \bbl@read@ini{##1}0%
4015       \endinput          % babel- .tex may contain onlypreamble's
4016     \endgroup}%           boxed, to avoid extra spaces:
4017   {\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.

```
4018 \def\bbl@setdigits#1#2#3#4#5{%
4019   \bbl@exp{%
4020     \def\<\languagename digits>####1{%          ie, \langdigits
4021       \<bbl@digits@\languagename>####1\\\@nil}%
4022     \let\<bbl@cntr@digits@\languagename>\<\languagename digits>%
4023     \def\<\languagename counter>####1{%      ie, \langcounter
4024       \\\expandafter\<bbl@counter@\languagename>%
4025       \\\csname c@####1\endcsname}%
4026     \def\<bbl@counter@\languagename>####1{% ie, \bbl@counter@lang
4027       \\\expandafter\<bbl@digits@\languagename>%
4028       \\\number####1\\\@nil}}%
4029   \def\bbl@tempa##1##2##3##4##5{%
4030     \bbl@exp{%     Wow, quite a lot of hashes! :-(
4031       \def\<bbl@digits@\languagename>########1{%
4032         \\\ifx########1\\\@nil              % ie, \bbl@digits@lang
4033         \\\else
4034           \\\ifx0########1#1%
4035           \\\else\\\ifx1########1#2%
4036           \\\else\\\ifx2########1#3%
4037           \\\else\\\ifx3########1#4%
4038           \\\else\\\ifx4########1#5%
4039           \\\else\\\ifx5########1##1%
4040           \\\else\\\ifx6########1##2%
4041           \\\else\\\ifx7########1##3%
4042           \\\else\\\ifx8########1##4%
4043           \\\else\\\ifx9########1##5%
4044           \\\else########1%
4045           \\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi
4046           \\\expandafter\<bbl@digits@\languagename>%
4047         \\\fi}}}%
4048   \bbl@tempa}
```

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

```
4049 \def\bbl@buildifcase#1 {% Returns \bbl@tempa, requires \toks@={}
4050   \ifx\\#1%                % \\ before, in case #1 is multiletter
4051     \bbl@exp{%
4052       \def\\\bbl@tempa####1{%
4053         \<ifcase>####1\space\the\toks@\<else>\\\@ctrerr\<fi>}}%
4054   \else
4055     \toks@\expandafter{\the\toks@\or #1}%
4056     \expandafter\bbl@buildifcase
4057   \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).

```
4058 \newcommand\localenumeral[2]{\bbl@cs{cntr@#1@\languagename}{#2}}
4059 \def\bbl@localecntr#1#2{\localenumeral{#2}{#1}}
4060 \newcommand\localecounter[2]{%
4061   \expandafter\bbl@localecntr
4062   \expandafter{\number\csname c@#2\endcsname}{#1}}
4063 \def\bbl@alphnumeral#1#2{%
4064   \expandafter\bbl@alphnumeral@i\number#2 76543210\@@{#1}}
4065 \def\bbl@alphnumeral@i#1#2#3#4#5#6#7#8\@@#9{%
4066   \ifcase\@car#8\@nil\or   % Currenty <10000, but prepared for bigger
4067     \bbl@alphnumeral@ii{#9}000000#1\or
4068     \bbl@alphnumeral@ii{#9}00000#1#2\or
```

```
4069     \bbl@alphnumeral@ii{#9}0000#1#2#3\or
4070     \bbl@alphnumeral@ii{#9}000#1#2#3#4\else
4071     \bbl@alphnum@invalid{>9999}%
4072   \fi}
4073 \def\bbl@alphnumeral@ii#1#2#3#4#5#6#7#8{%
4074   \bbl@ifunset{bbl@cntr@#1.F.\number#5#6#7#8@\languagename}%
4075     {\bbl@cs{cntr@#1.4@\languagename}#5%
4076      \bbl@cs{cntr@#1.3@\languagename}#6%
4077      \bbl@cs{cntr@#1.2@\languagename}#7%
4078      \bbl@cs{cntr@#1.1@\languagename}#8%
4079      \ifnum#6#7#8>\z@ % TODO. An ad hoc rule for Greek. Ugly.
4080        \bbl@ifunset{bbl@cntr@#1.S.321@\languagename}{}%
4081          {\bbl@cs{cntr@#1.S.321@\languagename}}%
4082      \fi}%
4083     {\bbl@cs{cntr@#1.F.\number#5#6#7#8@\languagename}}}
4084 \def\bbl@alphnum@invalid#1{%
4085   \bbl@error{Alphabetic numeral too large (#1)}%
4086     {Currently this is the limit.}}
```

The information in the identification section can be useful, so the following macro just exposes it
with a user command.

```
4087 \newcommand\localeinfo[1]{%
4088   \bbl@ifunset{bbl@\csname bbl@info@#1\endcsname @\languagename}%
4089     {\bbl@error{I've found no info for the current locale.\\%
4090                 The corresponding ini file has not been loaded\\%
4091                 Perhaps it doesn't exist}%
4092                {See the manual for details.}}%
4093     {\bbl@cs{\csname bbl@info@#1\endcsname @\languagename}}}
4094 % \@namedef{bbl@info@name.locale}{lcname}
4095 \@namedef{bbl@info@tag.ini}{lini}
4096 \@namedef{bbl@info@name.english}{elname}
4097 \@namedef{bbl@info@name.opentype}{lname}
4098 \@namedef{bbl@info@tag.bcp47}{tbcp}
4099 \@namedef{bbl@info@language.tag.bcp47}{lbcp}
4100 \@namedef{bbl@info@tag.opentype}{lotf}
4101 \@namedef{bbl@info@script.name}{esname}
4102 \@namedef{bbl@info@script.name.opentype}{sname}
4103 \@namedef{bbl@info@script.tag.bcp47}{sbcp}
4104 \@namedef{bbl@info@script.tag.opentype}{sotf}
4105 \let\bbl@ensureinfo\@gobble
4106 \newcommand\BabelEnsureInfo{%
4107   \ifx\InputIfFileExists\@undefined\else
4108     \def\bbl@ensureinfo##1{%
4109       \bbl@ifunset{bbl@lname@##1}{\bbl@load@info{##1}}{}}%
4110   \fi
4111   \bbl@foreach\bbl@loaded{{%
4112     \def\languagename{##1}%
4113     \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.

```
4114 \newcommand\getlocaleproperty{%
4115   \@ifstar\bbl@getproperty@s\bbl@getproperty@x}
4116 \def\bbl@getproperty@s#1#2#3{%
4117   \let#1\relax
4118   \def\bbl@elt##1##2##3{%
4119     \bbl@ifsamestring{##1/##2}{#3}%
4120       {\providecommand#1{##3}%
```

```
4121        \def\bbl@elt####1####2####3{}}%
4122      {}}%
4123  \bbl@cs{inidata@#2}}%
4124 \def\bbl@getproperty@x#1#2#3{%
4125  \bbl@getproperty@s{#1}{#2}{#3}%
4126  \ifx#1\relax
4127    \bbl@error
4128      {Unknown key for locale '#2':\\%
4129       #3\\%
4130       \string#1 will be set to \relax}%
4131      {Perhaps you misspelled it.}%
4132  \fi}
4133 \let\bbl@ini@loaded\@empty
4134 \newcommand\LocaleForEach{\bbl@foreach\bbl@ini@loaded}
```

10 Adjusting the Babel bahavior

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

```
4135 \newcommand\babeladjust[1]{%  TODO. Error handling.
4136  \bbl@forkv{#1}{%
4137    \bbl@ifunset{bbl@ADJ@##1@##2}%
4138      {\bbl@cs{ADJ@##1}{##2}}%
4139      {\bbl@cs{ADJ@##1@##2}}}}
4140 %
4141 \def\bbl@adjust@lua#1#2{%
4142  \ifvmode
4143    \ifnum\currentgrouplevel=\z@
4144      \directlua{ Babel.#2 }%
4145      \expandafter\expandafter\expandafter\@gobble
4146    \fi
4147  \fi
4148  {\bbl@error   % The error is gobbled if everything went ok.
4149    {Currently, #1 related features can be adjusted only\\%
4150     in the main vertical list.}%
4151    {Maybe things change in the future, but this is what it is.}}}
4152 \@namedef{bbl@ADJ@bidi.mirroring@on}{%
4153  \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
4154 \@namedef{bbl@ADJ@bidi.mirroring@off}{%
4155  \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
4156 \@namedef{bbl@ADJ@bidi.text@on}{%
4157  \bbl@adjust@lua{bidi}{bidi_enabled=true}}
4158 \@namedef{bbl@ADJ@bidi.text@off}{%
4159  \bbl@adjust@lua{bidi}{bidi_enabled=false}}
4160 \@namedef{bbl@ADJ@bidi.mapdigits@on}{%
4161  \bbl@adjust@lua{bidi}{digits_mapped=true}}
4162 \@namedef{bbl@ADJ@bidi.mapdigits@off}{%
4163  \bbl@adjust@lua{bidi}{digits_mapped=false}}
4164 %
4165 \@namedef{bbl@ADJ@linebreak.sea@on}{%
4166  \bbl@adjust@lua{linebreak}{sea_enabled=true}}
4167 \@namedef{bbl@ADJ@linebreak.sea@off}{%
4168  \bbl@adjust@lua{linebreak}{sea_enabled=false}}
4169 \@namedef{bbl@ADJ@linebreak.cjk@on}{%
4170  \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
4171 \@namedef{bbl@ADJ@linebreak.cjk@off}{%
4172  \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
4173 \@namedef{bbl@ADJ@justify.arabic@on}{%
4174  \bbl@adjust@lua{linebreak}{arabic.justify_enabled=true}}
```

```
4175 \@namedef{bbl@ADJ@justify.arabic@off}{%
4176   \bbl@adjust@lua{linebreak}{arabic.justify_enabled=false}}
4177 %
4178 \def\bbl@adjust@layout#1{%
4179   \ifvmode
4180     #1%
4181     \expandafter\@gobble
4182   \fi
4183   {\bbl@error    % The error is gobbled if everything went ok.
4184     {Currently, layout related features can be adjusted only\\%
4185      in vertical mode.}%
4186    {Maybe things change in the future, but this is what it is.}}}
4187 \@namedef{bbl@ADJ@layout.tabular@on}{%
4188   \bbl@adjust@layout{\let\@tabular\bbl@NL@@tabular}}
4189 \@namedef{bbl@ADJ@layout.tabular@off}{%
4190   \bbl@adjust@layout{\let\@tabular\bbl@OL@@tabular}}
4191 \@namedef{bbl@ADJ@layout.lists@on}{%
4192   \bbl@adjust@layout{\let\list\bbl@NL@list}}
4193 \@namedef{bbl@ADJ@layout.lists@off}{%
4194   \bbl@adjust@layout{\let\list\bbl@OL@list}}
4195 \@namedef{bbl@ADJ@hyphenation.extra@on}{%
4196   \bbl@activateposthyphen}
4197 %
4198 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
4199   \bbl@bcpallowedtrue}
4200 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
4201   \bbl@bcpallowedfalse}
4202 \@namedef{bbl@ADJ@autoload.bcp47.prefix}#1{%
4203   \def\bbl@bcp@prefix{#1}}
4204 \def\bbl@bcp@prefix{bcp47-}
4205 \@namedef{bbl@ADJ@autoload.options}#1{%
4206   \def\bbl@autoload@options{#1}}
4207 \let\bbl@autoload@bcpoptions\@empty
4208 \@namedef{bbl@ADJ@autoload.bcp47.options}#1{%
4209   \def\bbl@autoload@bcpoptions{#1}}
4210 \newif\ifbbl@bcptoname
4211 \@namedef{bbl@ADJ@bcp47.toname@on}{%
4212   \bbl@bcptonametrue
4213   \BabelEnsureInfo}
4214 \@namedef{bbl@ADJ@bcp47.toname@off}{%
4215   \bbl@bcptonamefalse}
4216 \@namedef{bbl@ADJ@prehyphenation.disable@nohyphenation}{%
4217   \directlua{ Babel.ignore_pre_char = function(node)
4218     return (node.lang == \the\csname l@nohyphenation\endcsname)
4219   end }}
4220 \@namedef{bbl@ADJ@prehyphenation.disable@off}{%
4221   \directlua{ Babel.ignore_pre_char = function(node)
4222     return false
4223   end }}
4224 % TODO: use babel name, override
4225 %
4226 % As the final task, load the code for lua.
4227 %
4228 \ifx\directlua\@undefined\else
4229   \ifx\bbl@luapatterns\@undefined
4230     \input luababel.def
4231   \fi
4232 \fi
4233 ⟨/core⟩
```

156

A proxy file for switch.def

```
4234 ⟨*kernel⟩
4235 \let\bbl@onlyswitch\@empty
4236 \input babel.def
4237 \let\bbl@onlyswitch\@undefined
4238 ⟨/kernel⟩
4239 ⟨*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 can be used to include this code in the file hyphen.cfg. Code is written with lower level macros.

To make sure that LaTeX 2.09 executes the \@begindocumenthook we would want to alter \begin{document}, but as this done too often already, we add the new code at the front of \@preamblecmds. But we can only do that after it has been defined, so we add this piece of code to \dump.

This new definition starts by adding an instruction to write a message on the terminal and in the transcript file to inform the user of the preloaded hyphenation patterns.

Then everything is restored to the old situation and the format is dumped.

```
4240 ⟨⟨Make sure ProvidesFile is defined⟩⟩
4241 \ProvidesFile{hyphen.cfg}[⟨⟨date⟩⟩ ⟨⟨version⟩⟩ Babel hyphens]
4242 \xdef\bbl@format{\jobname}
4243 \def\bbl@version{⟨⟨version⟩⟩}
4244 \def\bbl@date{⟨⟨date⟩⟩}
4245 \ifx\AtBeginDocument\@undefined
4246   \def\@empty{}
4247   \let\orig@dump\dump
4248   \def\dump{%
4249     \ifx\@ztryfc\@undefined
4250     \else
4251       \toks0=\expandafter{\@preamblecmds}%
4252       \edef\@preamblecmds{\noexpand\@begindocumenthook\the\toks0}%
4253       \def\@begindocumenthook{}%
4254     \fi
4255     \let\dump\orig@dump\let\orig@dump\@undefined\dump}
4256 \fi
4257 ⟨⟨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.

```
4258 \def\process@line#1#2 #3 #4 {%
4259   \ifx=#1%
4260     \process@synonym{#2}%
4261   \else
4262     \process@language{#1#2}{#3}{#4}%
4263   \fi
4264   \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.

```
4265 \toks@{}
4266 \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.

```
4267 \def\process@synonym#1{%
4268   \ifnum\last@language=\m@ne
4269     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4270   \else
4271     \expandafter\chardef\csname l@#1\endcsname\last@language
4272     \wlog{\string\l@#1=\string\language\the\last@language}%
4273     \expandafter\let\csname #1hyphenmins\expandafter\endcsname
4274       \csname\languagename hyphenmins\endcsname
4275     \let\bbl@elt\relax
4276     \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}{}{}}%
4277   \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.

```
4278 \def\process@language#1#2#3{%
4279   \expandafter\addlanguage\csname l@#1\endcsname
4280   \expandafter\language\csname l@#1\endcsname
4281   \edef\languagename{#1}%
4282   \bbl@hook@everylanguage{#1}%
4283   % > luatex
4284   \bbl@get@enc#1::\@@@
4285   \begingroup
4286     \lefthyphenmin\m@ne
4287     \bbl@hook@loadpatterns{#2}%
4288     % > luatex
4289     \ifnum\lefthyphenmin=\m@ne
4290     \else
4291       \expandafter\xdef\csname #1hyphenmins\endcsname{%
4292         \the\lefthyphenmin\the\righthyphenmin}%
4293     \fi
4294   \endgroup
```

158

```
4295    \def\bbl@tempa{#3}%
4296    \ifx\bbl@tempa\@empty\else
4297      \bbl@hook@loadexceptions{#3}%
4298      % > luatex
4299    \fi
4300    \let\bbl@elt\relax
4301    \edef\bbl@languages{%
4302      \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
4303    \ifnum\the\language=\z@
4304      \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4305        \set@hyphenmins\tw@\thr@@\relax
4306      \else
4307        \expandafter\expandafter\expandafter\set@hyphenmins
4308          \csname #1hyphenmins\endcsname
4309      \fi
4310      \the\toks@
4311      \toks@{}%
4312    \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.

```
4313 \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.

```
4314 \def\bbl@hook@everylanguage#1{}
4315 \def\bbl@hook@loadpatterns#1{\input #1\relax}
4316 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
4317 \def\bbl@hook@loadkernel#1{%
4318   \def\addlanguage{\csname newlanguage\endcsname}%
4319   \def\adddialect##1##2{%
4320     \global\chardef##1##2\relax
4321     \wlog{\string##1 = a dialect from \string\language##2}}%
4322   \def\iflanguage##1{%
4323     \expandafter\ifx\csname l@##1\endcsname\relax
4324       \@nolanerr{##1}%
4325     \else
4326       \ifnum\csname l@##1\endcsname=\language
4327         \expandafter\expandafter\expandafter\@firstoftwo
4328       \else
4329         \expandafter\expandafter\expandafter\@secondoftwo
4330       \fi
4331     \fi}%
4332   \def\providehyphenmins##1##2{%
4333     \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
4334       \@namedef{##1hyphenmins}{##2}%
4335     \fi}%
4336   \def\set@hyphenmins##1##2{%
4337     \lefthyphenmin##1\relax
4338     \righthyphenmin##2\relax}%
4339   \def\selectlanguage{%
4340     \errhelp{Selecting a language requires a package supporting it}%
4341     \errmessage{Not loaded}}%
4342   \let\foreignlanguage\selectlanguage
4343   \let\otherlanguage\selectlanguage
4344   \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4345   \def\bbl@usehooks##1##2{}% TODO. Temporary!!
```

```
4346   \def\setlocale{%
4347     \errhelp{Find an armchair, sit down and wait}%
4348     \errmessage{Not yet available}}%
4349   \let\uselocale\setlocale
4350   \let\locale\setlocale
4351   \let\selectlocale\setlocale
4352   \let\localename\setlocale
4353   \let\textlocale\setlocale
4354   \let\textlanguage\setlocale
4355   \let\languagetext\setlocale}
4356 \begingroup
4357   \def\AddBabelHook#1#2{%
4358     \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
4359       \def\next{\toks1}%
4360     \else
4361       \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname####1}%
4362     \fi
4363     \next}
4364   \ifx\directlua\@undefined
4365     \ifx\XeTeXinputencoding\@undefined\else
4366       \input xebabel.def
4367     \fi
4368   \else
4369     \input luababel.def
4370   \fi
4371   \openin1 = babel-\bbl@format.cfg
4372   \ifeof1
4373   \else
4374     \input babel-\bbl@format.cfg\relax
4375   \fi
4376   \closein1
4377 \endgroup
4378 \bbl@hook@loadkernel{switch.def}
```

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

```
4379 \openin1 = language.dat
```

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

```
4380 \def\languagename{english}%
4381 \ifeof1
4382   \message{I couldn't find the file language.dat,\space
4383           I will try the file hyphen.tex}
4384   \input hyphen.tex\relax
4385   \chardef\l@english\z@
4386 \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.

```
4387   \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.

```
4388   \loop
4389     \endlinechar\m@ne
4390     \read1 to \bbl@line
4391     \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.

```
4392    \if T\ifeof1F\fi T\relax
4393      \ifx\bbl@line\@empty\else
4394        \edef\bbl@line{\bbl@line\space\space\space}%
4395        \expandafter\process@line\bbl@line\relax
4396      \fi
4397  \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.

```
4398    \begingroup
4399      \def\bbl@elt#1#2#3#4{%
4400        \global\language=#2\relax
4401        \gdef\languagename{#1}%
4402        \def\bbl@elt##1##2##3##4{}%
4403      \bbl@languages
4404    \endgroup
4405  \fi
4406  \closein1
```

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

```
4407  \if/\the\toks@/\else
4408    \errhelp{language.dat loads no language, only synonyms}
4409    \errmessage{Orphan language synonym}
4410  \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.

```
4411  \let\bbl@line\@undefined
4412  \let\process@line\@undefined
4413  \let\process@synonym\@undefined
4414  \let\process@language\@undefined
4415  \let\bbl@get@enc\@undefined
4416  \let\bbl@hyph@enc\@undefined
4417  \let\bbl@tempa\@undefined
4418  \let\bbl@hook@loadkernel\@undefined
4419  \let\bbl@hook@everylanguage\@undefined
4420  \let\bbl@hook@loadpatterns\@undefined
4421  \let\bbl@hook@loadexceptions\@undefined
4422  ⟨/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].

```
4423  ⟨⟨∗More package options⟩⟩ ≡
4424  \chardef\bbl@bidimode\z@
4425  \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4426  \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
4427  \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4428  \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4429  \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4430  \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4431  ⟨⟨/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.

```
4432 ⟨⟨∗Font selection⟩⟩ ≡
4433 \bbl@trace{Font handling with fontspec}
4434 \ifx\ExplSyntaxOn\@undefined\else
4435   \ExplSyntaxOn
4436   \catcode`\ =10
4437   \def\bbl@loadfontspec{%
4438     \usepackage{fontspec}%
4439     \expandafter
4440     \def\csname msg~text~>~fontspec/language-not-exist\endcsname##1##2##3##4{%
4441       Font '\l_fontspec_fontname_tl' is using the\\%
4442       default features for language '##1'.\\%
4443       That's usually fine, because many languages\\%
4444       require no specific features, but if the output is\\%
4445       not as expected, consider selecting another font.}
4446     \expandafter
4447     \def\csname msg~text~>~fontspec/no-script\endcsname##1##2##3##4{%
4448       Font '\l_fontspec_fontname_tl' is using the\\%
4449       default features for script '##2'.\\%
4450       That's not always wrong, but if the output is\\%
4451       not as expected, consider selecting another font.}}
4452   \ExplSyntaxOff
4453 \fi
4454 \@onlypreamble\babelfont
4455 \newcommand\babelfont[2][]{%  1=langs/scripts 2=fam
4456   \bbl@foreach{#1}{%
4457     \expandafter\ifx\csname date##1\endcsname\relax
4458       \IfFileExists{babel-##1.tex}%
4459         {\babelprovide{##1}}%
4460         {}%
4461     \fi}%
4462   \edef\bbl@tempa{#1}%
4463   \def\bbl@tempb{#2}%  Used by \bbl@bblfont
4464   \ifx\fontspec\@undefined
4465     \bbl@loadfontspec
4466   \fi
4467   \EnableBabelHook{babel-fontspec}% Just calls \bbl@switchfont
4468   \bbl@bblfont}
4469 \newcommand\bbl@bblfont[2][]{% 1=features 2=fontname, @font=rm|sf|tt
4470   \bbl@ifunset{\bbl@tempb family}%
4471     {\bbl@providefam{\bbl@tempb}}%
4472     {\bbl@exp{%
4473       \\\bbl@sreplace\<\bbl@tempb family >%
4474         {\@nameuse{\bbl@tempb default}}{\<\bbl@tempb default>}}}%
4475   % For the default font, just in case:
4476   \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
4477   \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
4478     {\bbl@csarg\edef{\bbl@tempb dflt@}{<>{#1}{#2}}% save bbl@rmdflt@
4479     \bbl@exp{%
4480       \let\<bbl@\bbl@tempb dflt@\languagename>\<bbl@\bbl@tempb dflt@>%
4481       \\\bbl@font@set\<bbl@\bbl@tempb dflt@\languagename>%
4482                   \<\bbl@tempb default>\<\bbl@tempb family>}}%
4483     {\bbl@foreach\bbl@tempa{% ie bbl@rmdflt@lang / *scrt
```

```
4484            \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:

```
4485 \def\bbl@providefam#1{%
4486   \bbl@exp{%
4487     \\\newcommand\<#1default>{}% Just define it
4488     \\\bbl@add@list\\\bbl@font@fams{#1}%
4489     \\\DeclareRobustCommand\<#1family>{%
4490       \\\not@math@alphabet\<#1family>\relax
4491       \\\fontfamily\<#1default>\\\selectfont}%
4492     \\\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.

```
4493 \def\bbl@nostdfont#1{%
4494   \bbl@ifunset{bbl@WFF@\f@family}%
4495     {\bbl@csarg\gdef{WFF@\f@family}{}%  Flag, to avoid dupl warns
4496      \bbl@infowarn{The current font is not a babel standard family:\\%
4497        #1%
4498        \fontname\font\\%
4499        There is nothing intrinsically wrong with this warning, and\\%
4500        you can ignore it altogether if you do not need these\\%
4501        families. But if they are used in the document, you should be\\%
4502        aware 'babel' will no set Script and Language for them, so\\%
4503        you may consider defining a new family with \string\babelfont.\\%
4504        See the manual for further details about \string\babelfont.\\%
4505        Reported}}
4506     {}}%
4507 \gdef\bbl@switchfont{%
4508   \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
4509   \bbl@exp{%  eg Arabic -> arabic
4510     \lowercase{\edef\\\bbl@tempa{\bbl@cl{sname}}}}%
4511   \bbl@foreach\bbl@font@fams{%
4512     \bbl@ifunset{bbl@##1dflt@\languagename}%      (1) language?
4513       {\bbl@ifunset{bbl@##1dflt@*\bbl@tempa}%      (2) from script?
4514          {\bbl@ifunset{bbl@##1dflt@}%              2=F - (3) from generic?
4515            {}%                                     123=F - nothing!
4516            {\bbl@exp{%                             3=T - from generic
4517               \global\let\<bbl@##1dflt@\languagename>%
4518                        \<bbl@##1dflt@>}}}%
4519          {\bbl@exp{%                               2=T - from script
4520             \global\let\<bbl@##1dflt@\languagename>%
4521                        \<bbl@##1dflt@*\bbl@tempa>}}}%
4522       {}}%                                         1=T - language, already defined
4523   \def\bbl@tempa{\bbl@nostdfont{}}%
4524   \bbl@foreach\bbl@font@fams{%       don't gather with prev for
4525     \bbl@ifunset{bbl@##1dflt@\languagename}%
4526       {\bbl@cs{famrst@##1}%
4527        \global\bbl@csarg\let{famrst@##1}\relax}%
4528       {\bbl@exp{% order is relevant. TODO: but sometimes wrong!
4529          \\\bbl@add\\\originalTeX{%
4530            \\\bbl@font@rst{\bbl@cl{##1dflt}}%
4531                           \<##1default>\<##1family>{##1}}%
4532          \\\bbl@font@set\<bbl@##1dflt@\languagename>% the main part!
4533                         \<##1default>\<##1family>}}}%
4534   \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.

```
4535 \ifx\f@family\@undefined\else    % if latex
4536  \ifcase\bbl@engine            % if pdftex
4537    \let\bbl@ckeckstdfonts\relax
4538  \else
4539    \def\bbl@ckeckstdfonts{%
4540      \begingroup
4541        \global\let\bbl@ckeckstdfonts\relax
4542        \let\bbl@tempa\@empty
4543        \bbl@foreach\bbl@font@fams{%
4544          \bbl@ifunset{bbl@##1dflt@}%
4545            {\@nameuse{##1family}%
4546             \bbl@csarg\gdef{WFF@\f@family}{}% Flag
4547             \bbl@exp{\\\bbl@add\\\bbl@tempa{* \<##1family>= \f@family\\\\%
4548                \space\space\fontname\font\\\\}}%
4549             \bbl@csarg\xdef{##1dflt@}{\f@family}%
4550             \expandafter\xdef\csname ##1default\endcsname{\f@family}}%
4551            {}}%
4552        \ifx\bbl@tempa\@empty\else
4553          \bbl@infowarn{The following font families will use the default\\%
4554            settings for all or some languages:\\%
4555            \bbl@tempa
4556            There is nothing intrinsically wrong with it, but\\%
4557            'babel' will no set Script and Language, which could\\%
4558             be relevant in some languages. If your document uses\\%
4559             these families, consider redefining them with \string\babelfont.\\%
4560            Reported}%
4561        \fi
4562      \endgroup}
4563  \fi
4564 \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.

```
4565 \def\bbl@font@set#1#2#3{% eg \bbl@rmdflt@lang \rmdefault \rmfamily
4566   \bbl@xin@{<>}{#1}%
4567   \ifin@
4568     \bbl@exp{\\\bbl@fontspec@set\\#1\expandafter\@gobbletwo#1\\#3}%
4569   \fi
4570   \bbl@exp{%              'Unprotected' macros return prev values
4571     \def\\#2{#1}%         eg, \rmdefault{\bbl@rmdflt@lang}
4572     \\\bbl@ifsamestring{#2}{\f@family}%
4573       {\\#3%
4574        \\\bbl@ifsamestring{\f@series}{\bfdefault}{\\\bfseries}{}%
4575        \let\\\bbl@tempa\relax}%
4576       {}}}
4577 %    TODO - next should be global?, but even local does its job. I'm
4578 %    still not sure -- must investigate:
4579 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
4580   \let\bbl@tempe\bbl@mapselect
4581   \let\bbl@mapselect\relax
4582   \let\bbl@temp@fam#4%      eg, '\rmfamily', to be restored below
4583   \let#4\@empty     %       Make sure \renewfontfamily is valid
4584   \bbl@exp{%
4585     \let\\\bbl@temp@pfam\<\bbl@stripslash#4\space>% eg, '\rmfamily '
4586     \<keys_if_exist:nnF>{fontspec-opentype}{Script/\bbl@cl{sname}}%
4587       {\\\newfontscript{\bbl@cl{sname}}{\bbl@cl{sotf}}}%
4588     \<keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@cl{lname}}%
```

```
4589        {\\\newfontlanguage{\bbl@cl{lname}}{\bbl@cl{lotf}}}%
4590      \\\renewfontfamily\\#4%
4591        [\bbl@cs{lsys@\languagename},#2]}{#3}% ie \bbl@exp{..}{#3}
4592    \begingroup
4593      #4%
4594      \xdef#1{\f@family}%      eg, \bbl@rmdflt@lang{FreeSerif(0)}
4595    \endgroup
4596    \let#4\bbl@temp@fam
4597    \bbl@exp{\let\<\bbl@stripslash#4\space>}\bbl@temp@pfam
4598    \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.

```
4599 \def\bbl@font@rst#1#2#3#4{%
4600    \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.

```
4601 \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 :-).

```
4602 \newcommand\babelFSstore[2][]{%
4603    \bbl@ifblank{#1}%
4604      {\bbl@csarg\def{sname@#2}{Latin}}%
4605      {\bbl@csarg\def{sname@#2}{#1}}%
4606    \bbl@provide@dirs{#2}%
4607    \bbl@csarg\ifnum{wdir@#2}>\z@
4608      \let\bbl@beforeforeign\leavevmode
4609      \EnableBabelHook{babel-bidi}%
4610    \fi
4611    \bbl@foreach{#2}{%
4612      \bbl@FSstore{##1}{rm}\rmdefault\bbl@save@rmdefault
4613      \bbl@FSstore{##1}{sf}\sfdefault\bbl@save@sfdefault
4614      \bbl@FSstore{##1}{tt}\ttdefault\bbl@save@ttdefault}}
4615 \def\bbl@FSstore#1#2#3#4{%
4616    \bbl@csarg\edef{#2default#1}{#3}%
4617    \expandafter\addto\csname extras#1\endcsname{%
4618      \let#4#3%
4619      \ifx#3\f@family
4620        \edef#3{\csname bbl@#2default#1\endcsname}%
4621        \fontfamily{#3}\selectfont
4622      \else
4623        \edef#3{\csname bbl@#2default#1\endcsname}%
4624      \fi}%
4625    \expandafter\addto\csname noextras#1\endcsname{%
4626      \ifx#3\f@family
4627        \fontfamily{#4}\selectfont
4628      \fi
4629      \let#3#4}}
4630 \let\bbl@langfeatures\@empty
4631 \def\babelFSfeatures{% make sure \fontspec is redefined once
4632    \let\bbl@ori@fontspec\fontspec
4633    \renewcommand\fontspec[1][]{%
4634      \bbl@ori@fontspec[\bbl@langfeatures##1]}
4635    \let\babelFSfeatures\bbl@FSfeatures
4636    \babelFSfeatures}
4637 \def\bbl@FSfeatures#1#2{%
4638    \expandafter\addto\csname extras#1\endcsname{%
```

```
4639      \babel@save\bbl@langfeatures
4640      \edef\bbl@langfeatures{#2,}}}
4641 ⟨⟨/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.

```
4642 ⟨⟨∗Footnote changes⟩⟩ ≡
4643 \bbl@trace{Bidi footnotes}
4644 \ifnum\bbl@bidimode>\z@
4645   \def\bbl@footnote#1#2#3{%
4646     \@ifnextchar[%
4647       {\bbl@footnote@o{#1}{#2}{#3}}%
4648       {\bbl@footnote@x{#1}{#2}{#3}}}
4649   \long\def\bbl@footnote@x#1#2#3#4{%
4650     \bgroup
4651       \select@language@x{\bbl@main@language}%
4652       \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
4653     \egroup}
4654   \long\def\bbl@footnote@o#1#2#3[#4]#5{%
4655     \bgroup
4656       \select@language@x{\bbl@main@language}%
4657       \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
4658     \egroup}
4659   \def\bbl@footnotetext#1#2#3{%
4660     \@ifnextchar[%
4661       {\bbl@footnotetext@o{#1}{#2}{#3}}%
4662       {\bbl@footnotetext@x{#1}{#2}{#3}}}
4663   \long\def\bbl@footnotetext@x#1#2#3#4{%
4664     \bgroup
4665       \select@language@x{\bbl@main@language}%
4666       \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
4667     \egroup}
4668   \long\def\bbl@footnotetext@o#1#2#3[#4]#5{%
4669     \bgroup
4670       \select@language@x{\bbl@main@language}%
4671       \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
4672     \egroup}
4673   \def\BabelFootnote#1#2#3#4{%
4674     \ifx\bbl@fn@footnote\@undefined
4675       \let\bbl@fn@footnote\footnote
4676     \fi
4677     \ifx\bbl@fn@footnotetext\@undefined
4678       \let\bbl@fn@footnotetext\footnotetext
4679     \fi
4680     \bbl@ifblank{#2}%
4681       {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
4682        \@namedef{\bbl@stripslash#1text}%
4683          {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
4684       {\def#1{\bbl@exp{\\\bbl@footnote{\\\foreignlanguage{#2}}}{#3}{#4}}%
4685        \@namedef{\bbl@stripslash#1text}%
4686          {\bbl@exp{\\\bbl@footnotetext{\\\foreignlanguage{#2}}}{#3}{#4}}}}
4687 \fi
4688 ⟨⟨/Footnote changes⟩⟩
```

Now, the code.

```
4689 ⟨*xetex⟩
4690 \def\BabelStringsDefault{unicode}
4691 \let\xebbl@stop\relax
4692 \AddBabelHook{xetex}{encodedcommands}{%
4693   \def\bbl@tempa{#1}%
4694   \ifx\bbl@tempa\@empty
4695     \XeTeXinputencoding"bytes"%
4696   \else
4697     \XeTeXinputencoding"#1"%
4698   \fi
4699   \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4700 \AddBabelHook{xetex}{stopcommands}{%
4701   \xebbl@stop
4702   \let\xebbl@stop\relax}
4703 \def\bbl@intraspace#1 #2 #3\@@{%
4704   \bbl@csarg\gdef{xeisp@\languagename}%
4705     {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4706 \def\bbl@intrapenalty#1\@@{%
4707   \bbl@csarg\gdef{xeipn@\languagename}%
4708     {\XeTeXlinebreakpenalty #1\relax}}
4709 \def\bbl@provide@intraspace{%
4710   \bbl@xin@{/s}{/\bbl@cl{lnbrk}}%
4711   \ifin@\else\bbl@xin@{/c}{/\bbl@cl{lnbrk}}\fi
4712   \ifin@
4713     \bbl@ifunset{bbl@intsp@\languagename}{}%
4714       {\expandafter\ifx\csname bbl@intsp@\languagename\endcsname\@empty\else
4715         \ifx\bbl@KVP@intraspace\@nil
4716           \bbl@exp{%
4717             \\\bbl@intraspace\bbl@cl{intsp}\\\@@}%
4718         \fi
4719         \ifx\bbl@KVP@intrapenalty\@nil
4720           \bbl@intrapenalty0\@@
4721         \fi
4722       \fi
4723       \ifx\bbl@KVP@intraspace\@nil\else % We may override the ini
4724         \expandafter\bbl@intraspace\bbl@KVP@intraspace\@@
4725       \fi
4726       \ifx\bbl@KVP@intrapenalty\@nil\else
4727         \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
4728       \fi
4729       \bbl@exp{%
4730         \\\bbl@add\<extras\languagename>{%
4731           \XeTeXlinebreaklocale "\bbl@cl{tbcp}"%
4732           \<bbl@xeisp@\languagename>%
4733           \<bbl@xeipn@\languagename>}%
4734         \\\bbl@toglobal\<extras\languagename>%
4735         \\\bbl@add\<noextras\languagename>{%
4736           \XeTeXlinebreaklocale "en"}%
4737         \\\bbl@toglobal\<noextras\languagename>}%
4738       \ifx\bbl@ispacesize\@undefined
4739         \gdef\bbl@ispacesize{\bbl@cl{xeisp}}%
4740         \ifx\AtBeginDocument\@notprerr
4741           \expandafter\@secondoftwo  % to execute right now
4742         \fi
4743         \AtBeginDocument{%
4744           \expandafter\bbl@add
4745           \csname selectfont \endcsname{\bbl@ispacesize}%
```

167

```
4746        \expandafter\bbl@toglobal\csname selectfont \endcsname}%
4747      \fi}%
4748  \fi}
4749 \ifx\DisableBabelHook\@undefined\endinput\fi
4750 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4751 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
4752 \DisableBabelHook{babel-fontspec}
4753 ⟨⟨Font selection⟩⟩
4754 \input txtbabel.def
4755 ⟨/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.

```
4756 ⟨*texxet⟩
4757 \providecommand\bbl@provide@intraspace{}
4758 \bbl@trace{Redefinitions for bidi layout}
4759 \def\bbl@sspre@caption{%
4760   \bbl@exp{\everyhbox{\\\bbl@textdir\bbl@cs{wdir@\bbl@main@language}}}}
4761 \ifx\bbl@opt@layout\@nnil\endinput\fi  % No layout
4762 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
4763 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
4764 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
4765   \def\@hangfrom#1{%
4766     \setbox\@tempboxa\hbox{{#1}}%
4767     \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
4768     \noindent\box\@tempboxa}
4769   \def\raggedright{%
4770     \let\\\@centercr
4771     \bbl@startskip\z@skip
4772     \@rightskip\@flushglue
4773     \bbl@endskip\@rightskip
4774     \parindent\z@
4775     \parfillskip\bbl@startskip}
4776   \def\raggedleft{%
4777     \let\\\@centercr
4778     \bbl@startskip\@flushglue
4779     \bbl@endskip\z@skip
4780     \parindent\z@
4781     \parfillskip\bbl@endskip}
4782 \fi
4783 \IfBabelLayout{lists}
4784   {\bbl@sreplace\list
4785     {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
4786   \def\bbl@listleftmargin{%
4787     \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
4788   \ifcase\bbl@engine
4789     \def\labelenumii{)\theenumii(}% pdftex doesn't reverse ()
4790     \def\p@enumiii{\p@enumii)\theenumii(}%
4791   \fi
4792   \bbl@sreplace\@verbatim
4793     {\leftskip\@totalleftmargin}%
```

```
4794        {\bbl@startskip\textwidth
4795         \advance\bbl@startskip-\linewidth}%
4796      \bbl@sreplace\@verbatim
4797        {\rightskip\z@skip}%
4798        {\bbl@endskip\z@skip}}%
4799    {}
4800  \IfBabelLayout{contents}
4801    {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
4802     \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
4803    {}
4804  \IfBabelLayout{columns}
4805    {\bbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbl@outputhbox}%
4806     \def\bbl@outputhbox#1{%
4807       \hb@xt@\textwidth{%
4808         \hskip\columnwidth
4809         \hfil
4810         {\normalcolor\vrule \@width\columnseprule}%
4811         \hfil
4812         \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
4813         \hskip-\textwidth
4814         \hb@xt@\columnwidth{\box\@outputbox \hss}%
4815         \hskip\columnsep
4816         \hskip\columnwidth}}}%
4817    {}
4818  ⟨⟨Footnote changes⟩⟩
4819  \IfBabelLayout{footnotes}%
4820    {\BabelFootnote\footnote\languagename{}{}%
4821     \BabelFootnote\localfootnote\languagename{}{}%
4822     \BabelFootnote\mainfootnote{}{}{}}
4823    {}
```

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.

```
4824  \IfBabelLayout{counters}%
4825    {\let\bbl@latinarabic=\@arabic
4826     \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
4827     \let\bbl@asciiroman=\@roman
4828     \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciiroman#1}}}%
4829     \let\bbl@asciiRoman=\@Roman
4830     \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}}}{}
4831  ⟨/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).

```
4832 ⟨∗luatex⟩
4833 \ifx\AddBabelHook\@undefined % When plain.def, babel.sty starts
4834 \bbl@trace{Read language.dat}
4835 \ifx\bbl@readstream\@undefined
4836   \csname newread\endcsname\bbl@readstream
4837 \fi
4838 \begingroup
4839   \toks@{}
4840   \count@\z@ % 0=start, 1=0th, 2=normal
4841   \def\bbl@process@line#1#2 #3 #4 {%
4842     \ifx=#1%
4843       \bbl@process@synonym{#2}%
4844     \else
4845       \bbl@process@language{#1#2}{#3}{#4}%
4846     \fi
4847     \ignorespaces}
4848   \def\bbl@manylang{%
4849     \ifnum\bbl@last>\@ne
4850       \bbl@info{Non-standard hyphenation setup}%
4851     \fi
4852     \let\bbl@manylang\relax}
4853   \def\bbl@process@language#1#2#3{%
4854     \ifcase\count@
4855       \@ifundefined{zth@#1}{\count@\tw@}{\count@\@ne}%
4856     \or
4857       \count@\tw@
4858     \fi
4859     \ifnum\count@=\tw@
4860       \expandafter\addlanguage\csname l@#1\endcsname
4861       \language\allocationnumber
4862       \chardef\bbl@last\allocationnumber
4863       \bbl@manylang
4864       \let\bbl@elt\relax
4865       \xdef\bbl@languages{%
4866         \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
4867     \fi
4868     \the\toks@
4869     \toks@{}}
4870   \def\bbl@process@synonym@aux#1#2{%
4871     \global\expandafter\chardef\csname l@#1\endcsname#2\relax
```

```
4872      \let\bbl@elt\relax
4873      \xdef\bbl@languages{%
4874        \bbl@languages\bbl@elt{#1}{#2}{}{}}}%
4875    \def\bbl@process@synonym#1{%
4876      \ifcase\count@
4877        \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
4878      \or
4879        \@ifundefined{zth@#1}{\bbl@process@synonym@aux{#1}{0}}{}%
4880      \else
4881        \bbl@process@synonym@aux{#1}{\the\bbl@last}%
4882      \fi}
4883    \ifx\bbl@languages\@undefined % Just a (sensible?) guess
4884      \chardef\l@english\z@
4885      \chardef\l@USenglish\z@
4886      \chardef\bbl@last\z@
4887      \global\@namedef{bbl@hyphendata@0}{{hyphen.tex}{}}
4888      \gdef\bbl@languages{%
4889        \bbl@elt{english}{0}{hyphen.tex}{}%
4890        \bbl@elt{USenglish}{0}{}{}}
4891    \else
4892      \global\let\bbl@languages@format\bbl@languages
4893      \def\bbl@elt#1#2#3#4{% Remove all except language 0
4894        \ifnum#2>\z@\else
4895          \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
4896        \fi}%
4897      \xdef\bbl@languages{\bbl@languages}%
4898    \fi
4899    \def\bbl@elt#1#2#3#4{\@namedef{zth@#1}{}} % Define flags
4900    \bbl@languages
4901    \openin\bbl@readstream=language.dat
4902    \ifeof\bbl@readstream
4903      \bbl@warning{I couldn't find language.dat. No additional\\%
4904                   patterns loaded. Reported}%
4905    \else
4906      \loop
4907        \endlinechar\m@ne
4908        \read\bbl@readstream to \bbl@line
4909        \endlinechar`\^^M
4910        \if T\ifeof\bbl@readstream F\fi T\relax
4911          \ifx\bbl@line\@empty\else
4912            \edef\bbl@line{\bbl@line\space\space\space}%
4913            \expandafter\bbl@process@line\bbl@line\relax
4914          \fi
4915      \repeat
4916    \fi
4917  \endgroup
4918  \bbl@trace{Macros for reading patterns files}
4919  \def\bbl@get@enc#1:#2:#3\@@@{\def\bbl@hyph@enc{#2}}
4920  \ifx\babelcatcodetablenum\@undefined
4921    \ifx\newcatcodetable\@undefined
4922      \def\babelcatcodetablenum{5211}
4923      \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4924    \else
4925      \newcatcodetable\babelcatcodetablenum
4926      \newcatcodetable\bbl@pattcodes
4927    \fi
4928  \else
4929    \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4930  \fi
```

```
4931 \def\bbl@luapatterns#1#2{%
4932   \bbl@get@enc#1::\@@@
4933   \setbox\z@\hbox\bgroup
4934     \begingroup
4935       \savecatcodetable\babelcatcodetablenum\relax
4936       \initcatcodetable\bbl@pattcodes\relax
4937       \catcodetable\bbl@pattcodes\relax
4938         \catcode`\#=6  \catcode`\$=3 \catcode`\&=4 \catcode`\^=7
4939         \catcode`\_=8  \catcode`\{=1 \catcode`\}=2 \catcode`\~=13
4940         \catcode`\@=11 \catcode`\^^I=10 \catcode`\^^J=12
4941         \catcode`\<=12 \catcode`\>=12 \catcode`\*=12 \catcode`\.=12
4942         \catcode`\-=12 \catcode`\/=12 \catcode`\[=12 \catcode`\]=12
4943         \catcode`\`=12 \catcode`\'=12 \catcode`\"=12
4944         \input #1\relax
4945       \catcodetable\babelcatcodetablenum\relax
4946     \endgroup
4947     \def\bbl@tempa{#2}%
4948     \ifx\bbl@tempa\@empty\else
4949       \input #2\relax
4950     \fi
4951   \egroup}%
4952 \def\bbl@patterns@lua#1{%
4953   \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
4954     \csname l@#1\endcsname
4955     \edef\bbl@tempa{#1}%
4956   \else
4957     \csname l@#1:\f@encoding\endcsname
4958     \edef\bbl@tempa{#1:\f@encoding}%
4959   \fi\relax
4960   \@namedef{lu@texhyphen@loaded@\the\language}{}% Temp
4961   \@ifundefined{bbl@hyphendata@\the\language}%
4962     {\def\bbl@elt##1##2##3##4{%
4963        \ifnum##2=\csname l@\bbl@tempa\endcsname % #2=spanish, dutch:OT1...
4964          \def\bbl@tempb{##3}%
4965          \ifx\bbl@tempb\@empty\else % if not a synonymous
4966            \def\bbl@tempc{{##3}{##4}}%
4967          \fi
4968          \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
4969        \fi}%
4970      \bbl@languages
4971      \@ifundefined{bbl@hyphendata@\the\language}%
4972        {\bbl@info{No hyphenation patterns were set for\\%
4973                  language '\bbl@tempa'. Reported}}%
4974        {\expandafter\expandafter\expandafter\bbl@luapatterns
4975          \csname bbl@hyphendata@\the\language\endcsname}}{}}
4976 \endinput\fi
4977   % Here ends \ifx\AddBabelHook\@undefined
4978   % A few lines are only read by hyphen.cfg
4979 \ifx\DisableBabelHook\@undefined
4980   \AddBabelHook{luatex}{everylanguage}{%
4981     \def\process@language##1##2##3{%
4982       \def\process@line####1####2 ####3 ####4 {}}}
4983   \AddBabelHook{luatex}{loadpatterns}{%
4984     \input #1\relax
4985     \expandafter\gdef\csname bbl@hyphendata@\the\language\endcsname
4986       {{#1}{}}}
4987   \AddBabelHook{luatex}{loadexceptions}{%
4988     \input #1\relax
4989     \def\bbl@tempb##1##2{{##1}{#1}}%
```

172

```
4990      \expandafter\xdef\csname bbl@hyphendata@\the\language\endcsname
4991        {\expandafter\expandafter\expandafter\bbl@tempb
4992         \csname bbl@hyphendata@\the\language\endcsname}}
4993 \endinput\fi
4994   % Here stops reading code for hyphen.cfg
4995   % The following is read the 2nd time it's loaded
4996 \begingroup  % TODO - to a lua file
4997 \catcode`\%=12
4998 \catcode`\'=12
4999 \catcode`\"=12
5000 \catcode`\:=12
5001 \directlua{
5002   Babel = Babel or {}
5003   function Babel.bytes(line)
5004     return line:gsub("(.)",
5005       function (chr) return unicode.utf8.char(string.byte(chr)) end)
5006   end
5007   function Babel.begin_process_input()
5008     if luatexbase and luatexbase.add_to_callback then
5009       luatexbase.add_to_callback('process_input_buffer',
5010                                  Babel.bytes,'Babel.bytes')
5011     else
5012       Babel.callback = callback.find('process_input_buffer')
5013       callback.register('process_input_buffer',Babel.bytes)
5014     end
5015   end
5016   function Babel.end_process_input ()
5017     if luatexbase and luatexbase.remove_from_callback then
5018       luatexbase.remove_from_callback('process_input_buffer','Babel.bytes')
5019     else
5020       callback.register('process_input_buffer',Babel.callback)
5021     end
5022   end
5023   function Babel.addpatterns(pp, lg)
5024     local lg = lang.new(lg)
5025     local pats = lang.patterns(lg) or ''
5026     lang.clear_patterns(lg)
5027     for p in pp:gmatch('[^%s]+') do
5028       ss = ''
5029       for i in string.utfcharacters(p:gsub('%d', '')) do
5030         ss = ss .. '%d?' .. i
5031       end
5032       ss = ss:gsub('^%%d%?%.', '%%.') .. '%d?'
5033       ss = ss:gsub('%.%%d%?$', '%%.')
5034       pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
5035       if n == 0 then
5036         tex.sprint(
5037           [[\string\csname\space bbl@info\endcsname{New pattern: ]]
5038           .. p .. [[}]])
5039         pats = pats .. ' ' .. p
5040       else
5041         tex.sprint(
5042           [[\string\csname\space bbl@info\endcsname{Renew pattern: ]]
5043           .. p .. [[}]])
5044       end
5045     end
5046     lang.patterns(lg, pats)
5047   end
5048 }
```

```
5049 \endgroup
5050 \ifx\newattribute\@undefined\else
5051   \newattribute\bbl@attr@locale
5052   \directlua{ Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale'}
5053   \AddBabelHook{luatex}{beforeextras}{%
5054     \setattribute\bbl@attr@locale\localeid}
5055 \fi
5056 \def\BabelStringsDefault{unicode}
5057 \let\luabbl@stop\relax
5058 \AddBabelHook{luatex}{encodedcommands}{%
5059   \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
5060   \ifx\bbl@tempa\bbl@tempb\else
5061     \directlua{Babel.begin_process_input()}%
5062     \def\luabbl@stop{%
5063       \directlua{Babel.end_process_input()}}%
5064   \fi}%
5065 \AddBabelHook{luatex}{stopcommands}{%
5066   \luabbl@stop
5067   \let\luabbl@stop\relax}
5068 \AddBabelHook{luatex}{patterns}{%
5069   \@ifundefined{bbl@hyphendata@\the\language}%
5070     {\def\bbl@elt##1##2##3##4{%
5071       \ifnum##2=\csname l@##2\endcsname % #2=spanish, dutch:OT1...
5072         \def\bbl@tempb{##3}%
5073         \ifx\bbl@tempb\@empty\else % if not a synonymous
5074           \def\bbl@tempc{{##3}{##4}}%
5075         \fi
5076         \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5077       \fi}%
5078     \bbl@languages
5079     \@ifundefined{bbl@hyphendata@\the\language}%
5080       {\bbl@info{No hyphenation patterns were set for\\%
5081                 language '#2'. Reported}}%
5082       {\expandafter\expandafter\expandafter\bbl@luapatterns
5083         \csname bbl@hyphendata@\the\language\endcsname}}{}%
5084   \@ifundefined{bbl@patterns@}{}{%
5085     \begingroup
5086       \bbl@xin@{,\number\language,}{,\bbl@pttnlist}%
5087       \ifin@\else
5088         \ifx\bbl@patterns@\@empty\else
5089           \directlua{ Babel.addpatterns(
5090             [[\bbl@patterns@]], \number\language) }%
5091         \fi
5092         \@ifundefined{bbl@patterns@#1}%
5093           \@empty
5094           {\directlua{ Babel.addpatterns(
5095             [[\space\csname bbl@patterns@#1\endcsname]],
5096             \number\language) }}%
5097         \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
5098       \fi
5099     \endgroup}%
5100   \bbl@exp{%
5101     \bbl@ifunset{bbl@prehc@\languagename}{}%
5102       {\\\bbl@ifblank{\bbl@cs{prehc@\languagename}}{}%
5103         {\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.

```
5104 \@onlypreamble\babelpatterns
5105 \AtEndOfPackage{%
5106   \newcommand\babelpatterns[2][\@empty]{%
5107     \ifx\bbl@patterns@\relax
5108       \let\bbl@patterns@\@empty
5109     \fi
5110     \ifx\bbl@pttnlist\@empty\else
5111       \bbl@warning{%
5112         You must not intermingle \string\selectlanguage\space and\\%
5113         \string\babelpatterns\space or some patterns will not\\%
5114         be taken into account. Reported}%
5115     \fi
5116     \ifx\@empty#1%
5117       \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
5118     \else
5119       \edef\bbl@tempb{\zap@space#1 \@empty}%
5120       \bbl@for\bbl@tempa\bbl@tempb{%
5121         \bbl@fixname\bbl@tempa
5122         \bbl@iflanguage\bbl@tempa{%
5123           \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
5124             \@ifundefined{bbl@patterns@\bbl@tempa}%
5125               \@empty
5126               {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
5127           #2}}}%
5128     \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.

```
5129 % TODO - to a lua file
5130 \directlua{
5131   Babel = Babel or {}
5132   Babel.linebreaking = Babel.linebreaking or {}
5133   Babel.linebreaking.before = {}
5134   Babel.linebreaking.after = {}
5135   Babel.locale = {} % Free to use, indexed by \localeid
5136   function Babel.linebreaking.add_before(func)
5137     tex.print([[\noexpand\csname bbl@luahyphenate\endcsname]])
5138     table.insert(Babel.linebreaking.before, func)
5139   end
5140   function Babel.linebreaking.add_after(func)
5141     tex.print([[\noexpand\csname bbl@luahyphenate\endcsname]])
5142     table.insert(Babel.linebreaking.after, func)
5143   end
5144 }
5145 \def\bbl@intraspace#1 #2 #3\@@{%
5146   \directlua{
5147     Babel = Babel or {}
5148     Babel.intraspaces = Babel.intraspaces or {}
5149     Babel.intraspaces['\csname bbl@sbcp@\languagename\endcsname'] = %
5150       {b = #1, p = #2, m = #3}
5151     Babel.locale_props[\the\localeid].intraspace = %
5152       {b = #1, p = #2, m = #3}
5153   }}
5154 \def\bbl@intrapenalty#1\@@{%
```

```
5155  \directlua{
5156    Babel = Babel or {}
5157    Babel.intrapenalties = Babel.intrapenalties or {}
5158    Babel.intrapenalties['\csname bbl@sbcp@\languagename\endcsname'] = #1
5159    Babel.locale_props[\the\localeid].intrapenalty = #1
5160  }}
5161 \begingroup
5162 \catcode`\%=12
5163 \catcode`\^=14
5164 \catcode`\'=12
5165 \catcode`\~=12
5166 \gdef\bbl@seaintraspace{^
5167   \let\bbl@seaintraspace\relax
5168   \directlua{
5169    Babel = Babel or {}
5170    Babel.sea_enabled = true
5171    Babel.sea_ranges = Babel.sea_ranges or {}
5172    function Babel.set_chranges (script, chrng)
5173      local c = 0
5174      for s, e in string.gmatch(chrng..' ', '(.-)%.%.(.-)%s') do
5175        Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
5176        c = c + 1
5177      end
5178    end
5179    function Babel.sea_disc_to_space (head)
5180      local sea_ranges = Babel.sea_ranges
5181      local last_char = nil
5182      local quad = 655360        ^% 10 pt = 655360 = 10 * 65536
5183      for item in node.traverse(head) do
5184        local i = item.id
5185        if i == node.id'glyph' then
5186          last_char = item
5187        elseif i == 7 and item.subtype == 3 and last_char
5188            and last_char.char > 0x0C99 then
5189          quad = font.getfont(last_char.font).size
5190          for lg, rg in pairs(sea_ranges) do
5191            if last_char.char > rg[1] and last_char.char < rg[2] then
5192              lg = lg:sub(1, 4)  ^% Remove trailing number of, eg, Cyrl1
5193              local intraspace = Babel.intraspaces[lg]
5194              local intrapenalty = Babel.intrapenalties[lg]
5195              local n
5196              if intrapenalty ~= 0 then
5197                n = node.new(14, 0)      ^% penalty
5198                n.penalty = intrapenalty
5199                node.insert_before(head, item, n)
5200              end
5201              n = node.new(12, 13)      ^% (glue, spaceskip)
5202              node.setglue(n, intraspace.b * quad,
5203                              intraspace.p * quad,
5204                              intraspace.m * quad)
5205              node.insert_before(head, item, n)
5206              node.remove(head, item)
5207            end
5208          end
5209        end
5210      end
5211    end
5212  }^^
5213   \bbl@luahyphenate}
```

176

13.5 CJK line breaking

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

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

```
5214 \catcode`\%=14
5215 \gdef\bbl@cjkintraspace{%
5216   \let\bbl@cjkintraspace\relax
5217   \directlua{
5218     Babel = Babel or {}
5219     require('babel-data-cjk.lua')
5220     Babel.cjk_enabled = true
5221     function Babel.cjk_linebreak(head)
5222       local GLYPH = node.id'glyph'
5223       local last_char = nil
5224       local quad = 655360      % 10 pt = 655360 = 10 * 65536
5225       local last_class = nil
5226       local last_lang = nil
5227
5228       for item in node.traverse(head) do
5229         if item.id == GLYPH then
5230
5231           local lang = item.lang
5232
5233           local LOCALE = node.get_attribute(item,
5234                 luatexbase.registernumber'bbl@attr@locale')
5235           local props = Babel.locale_props[LOCALE]
5236
5237           local class = Babel.cjk_class[item.char].c
5238
5239           if class == 'cp' then class = 'cl' end % )] as CL
5240           if class == 'id' then class = 'I' end
5241
5242           local br = 0
5243           if class and last_class and Babel.cjk_breaks[last_class][class] then
5244             br = Babel.cjk_breaks[last_class][class]
5245           end
5246
5247           if br == 1 and props.linebreak == 'c' and
5248               lang ~= \the\l@nohyphenation\space and
5249               last_lang ~= \the\l@nohyphenation then
5250             local intrapenalty = props.intrapenalty
5251             if intrapenalty ~= 0 then
5252               local n = node.new(14, 0)     % penalty
5253               n.penalty = intrapenalty
5254               node.insert_before(head, item, n)
5255             end
5256             local intraspace = props.intraspace
5257             local n = node.new(12, 13)      % (glue, spaceskip)
5258             node.setglue(n, intraspace.b * quad,
5259                             intraspace.p * quad,
5260                             intraspace.m * quad)
5261             node.insert_before(head, item, n)
5262           end
5263
5264           if font.getfont(item.font) then
```

```
5265              quad = font.getfont(item.font).size
5266            end
5267            last_class = class
5268            last_lang = lang
5269          else % if penalty, glue or anything else
5270            last_class = nil
5271          end
5272        end
5273      lang.hyphenate(head)
5274    end
5275  }%
5276  \bbl@luahyphenate}
5277 \gdef\bbl@luahyphenate{%
5278  \let\bbl@luahyphenate\relax
5279  \directlua{
5280    luatexbase.add_to_callback('hyphenate',
5281    function (head, tail)
5282      if Babel.linebreaking.before then
5283        for k, func in ipairs(Babel.linebreaking.before)  do
5284          func(head)
5285        end
5286      end
5287      if Babel.cjk_enabled then
5288        Babel.cjk_linebreak(head)
5289      end
5290      lang.hyphenate(head)
5291      if Babel.linebreaking.after then
5292        for k, func in ipairs(Babel.linebreaking.after)  do
5293          func(head)
5294        end
5295      end
5296      if Babel.sea_enabled then
5297        Babel.sea_disc_to_space(head)
5298      end
5299    end,
5300    'Babel.hyphenate')
5301  }
5302 }
5303 \endgroup
5304 \def\bbl@provide@intraspace{%
5305  \bbl@ifunset{bbl@intsp@\languagename}{}%
5306    {\expandafter\ifx\csname bbl@intsp@\languagename\endcsname\@empty\else
5307      \bbl@xin@{/c}{/\bbl@cl{lnbrk}}%
5308      \ifin@          % cjk
5309        \bbl@cjkintraspace
5310        \directlua{
5311          Babel = Babel or {}
5312          Babel.locale_props = Babel.locale_props or {}
5313          Babel.locale_props[\the\localeid].linebreak = 'c'
5314        }%
5315        \bbl@exp{\\\bbl@intraspace\bbl@cl{intsp}\\\@@}%
5316        \ifx\bbl@KVP@intrapenalty\@nil
5317          \bbl@intrapenalty0\@@
5318        \fi
5319      \else            % sea
5320        \bbl@seaintraspace
5321        \bbl@exp{\\\bbl@intraspace\bbl@cl{intsp}\\\@@}%
5322        \directlua{
5323          Babel = Babel or {}
```

```
5324              Babel.sea_ranges = Babel.sea_ranges or {}
5325              Babel.set_chranges('\bbl@cl{sbcp}',
5326                                  '\bbl@cl{chrng}')
5327          }%
5328          \ifx\bbl@KVP@intrapenalty\@nil
5329            \bbl@intrapenalty0\@@
5330          \fi
5331        \fi
5332      \fi
5333      \ifx\bbl@KVP@intrapenalty\@nil\else
5334        \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
5335      \fi}}
```

13.6   Arabic justification

```
5336 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5337 \def\bblar@chars{%
5338   0628,0629,062A,062B,062C,062D,062E,062F,0630,0631,0632,0633,%
5339   0634,0635,0636,0637,0638,0639,063A,063B,063C,063D,063E,063F,%
5340   0640,0641,0642,0643,0644,0645,0646,0647,0649}
5341 \def\bblar@elongated{%
5342   0626,0628,062A,062B,0633,0634,0635,0636,063B,%
5343   063C,063D,063E,063F,0641,0642,0643,0644,0646,%
5344   0649,064A}
5345 \begingroup
5346   \catcode`\_=11 \catcode`:=11
5347   \gdef\bblar@nofswarn{\gdef\msg_warning:nnx##1##2##3{}}
5348 \endgroup
5349 \gdef\bbl@arabicjust{%
5350   \let\bbl@arabicjust\relax
5351   \newattribute\bblar@kashida
5352   \bblar@kashida=\z@
5353   \expandafter\bbl@add\csname selectfont \endcsname{{\bbl@parsejalt}}%
5354   \directlua{
5355     Babel.arabic.elong_map   = Babel.arabic.elong_map or {}
5356     Babel.arabic.elong_map[\the\localeid]   = {}
5357     luatexbase.add_to_callback('post_linebreak_filter',
5358       Babel.arabic.justify, 'Babel.arabic.justify')
5359   }}%
5360 % Save both node lists to make replacement. TODO. Save also widths to
5361 % make computations
5362 \def\bblar@fetchjalt#1#2#3#4{%
5363   \bbl@exp{\\\bbl@foreach{#1}}{%
5364     \bbl@ifunset{bblar@JE@##1}%
5365       {\setbox\z@\hbox{^^^^200d\char"##1#2}}%
5366       {\setbox\z@\hbox{^^^^200d\char"\@nameuse{bblar@JE@##1}#2}}%
5367     \directlua{%
5368       local last = nil
5369       for item in node.traverse(tex.box[0].head) do
5370         if item.id == node.id'glyph' and item.char > 0x600 and
5371            not (item.char == 0x200D) then
5372           last = item
5373         end
5374       end
5375       Babel.arabic.#3['##1#4'] = last.char
5376     }}}
5377 % Brute force. No rules at all, yet. The ideal: look at jalt table. And
5378 % perhaps other tables (falt?, cswh?). What about kaf? And diacritic
5379 % positioning?
```

```
5380 \gdef\bbl@parsejalt{%
5381   \ifx\addfontfeature\@undefined\else
5382     \bbl@xin@{/e}{/\bbl@cl{lnbrk}}%
5383     \ifin@
5384       \directlua{%
5385         if Babel.arabic.elong_map[\the\localeid][\fontid\font] == nil then
5386           Babel.arabic.elong_map[\the\localeid][\fontid\font] = {}
5387           tex.print([[\string\csname\space bbl@parsejalti\endcsname]])
5388         end
5389       }%
5390     \fi
5391   \fi}
5392 \gdef\bbl@parsejalti{%
5393   \begingroup
5394     \let\bbl@parsejalt\relax      % To avoid infinite loop
5395     \edef\bbl@tempb{\fontid\font}%
5396     \bblar@nofswarn
5397     \bblar@fetchjalt\bblar@elongated{}{from}{}%
5398     \bblar@fetchjalt\bblar@chars{^^^^064a}{from}{a}% Alef maksura
5399     \bblar@fetchjalt\bblar@chars{^^^^0649}{from}{y}% Yeh
5400     \addfontfeature{RawFeature=+jalt}%
5401     % \@namedef{bblar@JE@0643}{06AA}% todo: catch medial kaf
5402     \bblar@fetchjalt\bblar@elongated{}{dest}{}%
5403     \bblar@fetchjalt\bblar@chars{^^^^064a}{dest}{a}%
5404     \bblar@fetchjalt\bblar@chars{^^^^0649}{dest}{y}%
5405     \directlua{%
5406       for k, v in pairs(Babel.arabic.from) do
5407         if Babel.arabic.dest[k] and
5408             not (Babel.arabic.from[k] == Babel.arabic.dest[k]) then
5409           Babel.arabic.elong_map[\the\localeid][\bbl@tempb]
5410             [Babel.arabic.from[k]] = Babel.arabic.dest[k]
5411         end
5412       end
5413     }%
5414   \endgroup}
5415 %
5416 \begingroup
5417 \catcode`\#=11
5418 \catcode`\~=11
5419 \directlua{
5420
5421 Babel.arabic = Babel.arabic or {}
5422 Babel.arabic.from = {}
5423 Babel.arabic.dest = {}
5424 Babel.arabic.justify_factor = 0.95
5425 Babel.arabic.justify_enabled = true
5426
5427 function Babel.arabic.justify(head)
5428   if not Babel.arabic.justify_enabled then return head end
5429   local d, new
5430   local k_list, k_item, pos_inline
5431   local width, width_new, full, k_curr, wt_pos, goal
5432   local subst_done = false
5433   local elong_map = Babel.arabic.elong_map
5434   local last_line
5435   local GLYPH = node.id'glyph'
5436   local KASHIDA = luatexbase.registernumber'bblar@kashida'
5437   local LOCALE = luatexbase.registernumber'bbl@attr@locale'
5438
```

```
5439  for line in node.traverse_id(node.id'hlist', head) do
5440    % Exclude last line. todo. But-- it discards one-word lines, too!
5441    if (line.glue_sign == 1 and line.glue_order == 0) then
5442      elongs = {}     % Stores elongated candidates of each line
5443      k_list = {}     % And all letters with kashida
5444      pos_inline = 0  % Not yet used
5445
5446      for n in node.traverse_id(GLYPH, line.head) do
5447        pos_inline = pos_inline + 1 % To find where it is. Not used.
5448
5449        % Elongated glyphs
5450        if elong_map then
5451          local locale = node.get_attribute(n, LOCALE)
5452          if elong_map[locale] and elong_map[locale][n.font] and
5453              elong_map[locale][n.font][n.char] then
5454            table.insert(elongs, {node = n, locale = locale} )
5455            node.set_attribute(n.prev, KASHIDA, 0)
5456          end
5457        end
5458
5459        % Tatwil
5460        if Babel.kashida_wts then
5461          local k_wt = node.get_attribute(n, KASHIDA)
5462          if k_wt > 0 then
5463            table.insert(k_list, {node = n, weight = k_wt, pos = pos_inline})
5464          end
5465        end
5466
5467      end % of node.traverse_id
5468
5469      if #elongs == 0 and #k_list == 0 then goto next_line end
5470
5471      full = line.width
5472      goal = full * Babel.arabic.justify_factor % A bit crude
5473      width = node.dimensions(line.head)    % The 'natural' width
5474
5475      % == Elongated ==
5476      % Original idea taken from 'chikenize'
5477      while (#elongs > 0 and width < goal) do
5478        subst_done = true
5479        local x = #elongs
5480        local curr = elongs[x].node
5481        local oldchar = curr.char
5482        curr.char = elong_map[elongs[x].locale][curr.font][curr.char]
5483        width = node.dimensions(line.head)  % Check if the line is too wide
5484        % Substitute back if the line would be too wide and break:
5485        if width > goal then
5486          curr.char = oldchar
5487          break
5488        end
5489        % If continue, pop the just substituted node from the list:
5490        table.remove(elongs, x)
5491      end
5492
5493      % == Tatwil ==
5494      if #k_list == 0 then goto next_line end
5495
5496      width = node.dimensions(line.head)    % The 'natural' width
5497      k_curr = #k_list
```

```
5498        wt_pos = 1
5499
5500      while width < goal do
5501        subst_done = true
5502        k_item = k_list[k_curr].node
5503        if k_list[k_curr].weight == Babel.kashida_wts[wt_pos] then
5504          d = node.copy(k_item)
5505          d.char = 0x0640
5506          line.head, new = node.insert_after(line.head, k_item, d)
5507          width_new = node.dimensions(line.head)
5508          if width > goal or width == width_new then
5509            node.remove(line.head, new) % Better compute before
5510            break
5511          end
5512          width = width_new
5513        end
5514        if k_curr == 1 then
5515          k_curr = #k_list
5516          wt_pos = (wt_pos >= table.getn(Babel.kashida_wts)) and 1 or wt_pos+1
5517        else
5518          k_curr = k_curr - 1
5519        end
5520      end
5521
5522      ::next_line::
5523
5524      % Must take into account marks and ins, see luatex manual.
5525      % Have to be executed only if there are changes. Investigate
5526      % what's going on exactly.
5527      if subst_done then
5528        node.insert_before(head, line, node.hpack(line.head, full, 'exactly'))
5529        node.remove(head, line)
5530      end
5531    end % if process line
5532  end % for lines
5533  return head
5534 end
5535 }
5536 \endgroup
5537 \fi\fi % Arabic just block
```

13.7 Common stuff

```
5538 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
5539 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
5540 \DisableBabelHook{babel-fontspec}
5541 ⟨⟨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.

```
5542 % TODO - to a lua file
5543 \directlua{
5544 Babel.script_blocks = {
```

```
5545    ['dflt'] = {},
5546    ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
5547                {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
5548    ['Armn'] = {{0x0530, 0x058F}},
5549    ['Beng'] = {{0x0980, 0x09FF}},
5550    ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
5551    ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
5552    ['Cyrl'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
5553                {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
5554    ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
5555    ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
5556                {0xAB00, 0xAB2F}},
5557    ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
5558    % Don't follow strictly Unicode, which places some Coptic letters in
5559    % the 'Greek and Coptic' block
5560    ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
5561    ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
5562                {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
5563                {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
5564                {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
5565                {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
5566                {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
5567    ['Hebr'] = {{0x0590, 0x05FF}},
5568    ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
5569                {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
5570    ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
5571    ['Knda'] = {{0x0C80, 0x0CFF}},
5572    ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
5573                {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
5574                {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
5575    ['Laoo'] = {{0x0E80, 0x0EFF}},
5576    ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
5577                {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
5578                {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
5579    ['Mahj'] = {{0x11150, 0x1117F}},
5580    ['Mlym'] = {{0x0D00, 0x0D7F}},
5581    ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
5582    ['Orya'] = {{0x0B00, 0x0B7F}},
5583    ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
5584    ['Syrc'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
5585    ['Taml'] = {{0x0B80, 0x0BFF}},
5586    ['Telu'] = {{0x0C00, 0x0C7F}},
5587    ['Tfng'] = {{0x2D30, 0x2D7F}},
5588    ['Thai'] = {{0x0E00, 0x0E7F}},
5589    ['Tibt'] = {{0x0F00, 0x0FFF}},
5590    ['Vaii'] = {{0xA500, 0xA63F}},
5591    ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
5592 }
5593
5594 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
5595 Babel.script_blocks.Hant = Babel.script_blocks.Hans
5596 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
5597
5598 function Babel.locale_map(head)
5599   if not Babel.locale_mapped then return head end
5600
5601   local LOCALE = luatexbase.registernumber'bbl@attr@locale'
5602   local GLYPH = node.id('glyph')
5603   local inmath = false
```

183

```
5604    local toloc_save
5605    for item in node.traverse(head) do
5606      local toloc
5607      if not inmath and item.id == GLYPH then
5608        % Optimization: build a table with the chars found
5609        if Babel.chr_to_loc[item.char] then
5610          toloc = Babel.chr_to_loc[item.char]
5611        else
5612          for lc, maps in pairs(Babel.loc_to_scr) do
5613            for _, rg in pairs(maps) do
5614              if item.char >= rg[1] and item.char <= rg[2] then
5615                Babel.chr_to_loc[item.char] = lc
5616                toloc = lc
5617                break
5618              end
5619            end
5620          end
5621        end
5622        % Now, take action, but treat composite chars in a different
5623        % fashion, because they 'inherit' the previous locale. Not yet
5624        % optimized.
5625        if not toloc and
5626            (item.char >= 0x0300 and item.char <= 0x036F) or
5627            (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
5628            (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
5629          toloc = toloc_save
5630        end
5631        if toloc and toloc > -1 then
5632          if Babel.locale_props[toloc].lg then
5633            item.lang = Babel.locale_props[toloc].lg
5634            node.set_attribute(item, LOCALE, toloc)
5635          end
5636          if Babel.locale_props[toloc]['/'..item.font] then
5637            item.font = Babel.locale_props[toloc]['/'..item.font]
5638          end
5639          toloc_save = toloc
5640        end
5641      elseif not inmath and item.id == 7 then
5642        item.replace = item.replace and Babel.locale_map(item.replace)
5643        item.pre     = item.pre and Babel.locale_map(item.pre)
5644        item.post    = item.post and Babel.locale_map(item.post)
5645      elseif item.id == node.id'math' then
5646        inmath = (item.subtype == 0)
5647      end
5648    end
5649    return head
5650 end
5651 }
```

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

```
5652 \newcommand\babelcharproperty[1]{%
5653    \count@=#1\relax
5654    \ifvmode
5655      \expandafter\bbl@chprop
5656    \else
5657      \bbl@error{\string\babelcharproperty\space can be used only in\\%
5658                 vertical mode (preamble or between paragraphs)}%
5659                {See the manual for futher info}%
```

```
5660    \fi}
5661 \newcommand\bbl@chprop[3][\the\count@]{%
5662    \@tempcnta=#1\relax
5663    \bbl@ifunset{bbl@chprop@#2}%
5664      {\bbl@error{No property named '#2'. Allowed values are\\%
5665                  direction (bc), mirror (bmg), and linebreak (lb)}%
5666                  {See the manual for futher info}}%
5667      {}%
5668    \loop
5669      \bbl@cs{chprop@#2}{#3}%
5670    \ifnum\count@<\@tempcnta
5671      \advance\count@\@ne
5672    \repeat}
5673 \def\bbl@chprop@direction#1{%
5674    \directlua{
5675      Babel.characters[\the\count@] =  Babel.characters[\the\count@] or {}
5676      Babel.characters[\the\count@]['d'] = '#1'
5677    }}
5678 \let\bbl@chprop@bc\bbl@chprop@direction
5679 \def\bbl@chprop@mirror#1{%
5680    \directlua{
5681      Babel.characters[\the\count@] =  Babel.characters[\the\count@] or {}
5682      Babel.characters[\the\count@]['m'] = '\number#1'
5683    }}
5684 \let\bbl@chprop@bmg\bbl@chprop@mirror
5685 \def\bbl@chprop@linebreak#1{%
5686    \directlua{
5687      Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
5688      Babel.cjk_characters[\the\count@]['c'] = '#1'
5689    }}
5690 \let\bbl@chprop@lb\bbl@chprop@linebreak
5691 \def\bbl@chprop@locale#1{%
5692    \directlua{
5693      Babel.chr_to_loc = Babel.chr_to_loc or {}
5694      Babel.chr_to_loc[\the\count@] =
5695        \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@@#1}}\space
5696    }}
```

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).

After declaring the table containing the patterns with their replacements, we define some auxiliary functions: str_to_nodes converts the string returned by a function to a node list, taking the node at base as a model (font, language, etc.); fetch_word fetches a series of glyphs and discretionaries, which pattern is matched against (if there is a match, it is called again before trying other patterns, and this is very likely the main bottleneck).

post_hyphenate_replace is the callback applied after lang.hyphenate. This means the automatic hyphenation points are known. As empty captures return a byte position (as explained in the luatex manual), we must convert it to a utf8 position. With first, the last byte can be the leading byte in a utf8 sequence, so we just remove it and add 1 to the resulting length. With last we must take into account the capture position points to the next character. Here word_head points to the starting node of the text to be matched.

```
5697 \begingroup % TODO - to a lua file
5698 \catcode`\~=12
5699 \catcode`\#=12
5700 \catcode`\%=12
5701 \catcode`\&=14
5702 \directlua{
5703    Babel.linebreaking.replacements = {}
5704    Babel.linebreaking.replacements[0] = {}  &% pre
```

```
5705   Babel.linebreaking.replacements[1] = {}   &% post
5706
5707   &% Discretionaries contain strings as nodes
5708   function Babel.str_to_nodes(fn, matches, base)
5709     local n, head, last
5710     if fn == nil then return nil end
5711     for s in string.utfvalues(fn(matches)) do
5712       if base.id == 7 then
5713         base = base.replace
5714       end
5715       n = node.copy(base)
5716       n.char    = s
5717       if not head then
5718         head = n
5719       else
5720         last.next = n
5721       end
5722       last = n
5723     end
5724     return head
5725   end
5726
5727   Babel.fetch_subtext = {}
5728
5729   Babel.ignore_pre_char = function(node)
5730     return (node.lang == \the\l@nohyphenation)
5731   end
5732
5733   &% Merging both functions doesn't seen feasible, because there are too
5734   &% many differences.
5735   Babel.fetch_subtext[0] = function(head)
5736     local word_string = ''
5737     local word_nodes = {}
5738     local lang
5739     local item = head
5740     local inmath = false
5741
5742     while item do
5743
5744       if item.id == 11 then
5745         inmath = (item.subtype == 0)
5746       end
5747
5748       if inmath then
5749         &% pass
5750
5751       elseif item.id == 29 then
5752         local locale = node.get_attribute(item, Babel.attr_locale)
5753
5754         if lang == locale or lang == nil then
5755           lang = lang or locale
5756           if Babel.ignore_pre_char(item) then
5757             word_string = word_string .. Babel.us_char
5758           else
5759             word_string = word_string .. unicode.utf8.char(item.char)
5760           end
5761           word_nodes[#word_nodes+1] = item
5762         else
5763           break
```

```
5764            end
5765
5766        elseif item.id == 12 and item.subtype == 13 then
5767          word_string = word_string .. ' '
5768          word_nodes[#word_nodes+1] = item
5769
5770        &% Ignore leading unrecognized nodes, too.
5771        elseif word_string ~= '' then
5772          word_string = word_string .. Babel.us_char
5773          word_nodes[#word_nodes+1] = item  &% Will be ignored
5774        end
5775
5776        item = item.next
5777      end
5778
5779      &% Here and above we remove some trailing chars but not the
5780      &% corresponding nodes. But they aren't accessed.
5781      if word_string:sub(-1) == ' ' then
5782        word_string = word_string:sub(1,-2)
5783      end
5784      word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
5785      return word_string, word_nodes, item, lang
5786    end
5787
5788    Babel.fetch_subtext[1] = function(head)
5789      local word_string = ''
5790      local word_nodes = {}
5791      local lang
5792      local item = head
5793      local inmath = false
5794
5795      while item do
5796
5797        if item.id == 11 then
5798          inmath = (item.subtype == 0)
5799        end
5800
5801        if inmath then
5802          &% pass
5803
5804        elseif item.id == 29 then
5805          if item.lang == lang or lang == nil then
5806            if (item.char ~= 124) and (item.char ~= 61) then &% not =, not |
5807              lang = lang or item.lang
5808              word_string = word_string .. unicode.utf8.char(item.char)
5809              word_nodes[#word_nodes+1] = item
5810            end
5811          else
5812            break
5813          end
5814
5815        elseif item.id == 7 and item.subtype == 2 then
5816          word_string = word_string .. '='
5817          word_nodes[#word_nodes+1] = item
5818
5819        elseif item.id == 7 and item.subtype == 3 then
5820          word_string = word_string .. '|'
5821          word_nodes[#word_nodes+1] = item
5822
```

```
5823       &% (1) Go to next word if nothing was found, and (2) implictly
5824       &% remove leading USs.
5825       elseif word_string == '' then
5826         &% pass
5827
5828       &% This is the responsible for splitting by words.
5829       elseif (item.id == 12 and item.subtype == 13) then
5830         break
5831
5832       else
5833         word_string = word_string .. Babel.us_char
5834         word_nodes[#word_nodes+1] = item  &% Will be ignored
5835       end
5836
5837       item = item.next
5838     end
5839
5840   word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
5841   return word_string, word_nodes, item, lang
5842 end
5843
5844 function Babel.pre_hyphenate_replace(head)
5845   Babel.hyphenate_replace(head, 0)
5846 end
5847
5848 function Babel.post_hyphenate_replace(head)
5849   Babel.hyphenate_replace(head, 1)
5850 end
5851
5852 function Babel.debug_hyph(w, wn, sc, first, last, last_match)
5853   local ss = ''
5854   for pp = 1, 40 do
5855     if wn[pp] then
5856       if wn[pp].id == 29 then
5857         ss = ss .. unicode.utf8.char(wn[pp].char)
5858       else
5859         ss = ss .. '{' .. wn[pp].id .. '}'
5860       end
5861     end
5862   end
5863   print('nod', ss)
5864   print('lst_m',
5865     string.rep(' ', unicode.utf8.len(
5866         string.sub(w, 1, last_match))-1) .. '>')
5867   print('str', w)
5868   print('sc', string.rep(' ', sc-1) .. '^')
5869   if first == last then
5870     print('f=l', string.rep(' ', first-1) .. '!')
5871   else
5872     print('f/l', string.rep(' ', first-1) .. '[' ..
5873       string.rep(' ', last-first-1) .. ']')
5874   end
5875 end
5876
5877 Babel.us_char = string.char(31)
5878
5879 function Babel.hyphenate_replace(head, mode)
5880   local u = unicode.utf8
5881   local lbkr = Babel.linebreaking.replacements[mode]
```

```
5882
5883     local word_head = head
5884
5885     while true do  &% for each subtext block
5886
5887       local w, w_nodes, nw, lang = Babel.fetch_subtext[mode](word_head)
5888
5889       if Babel.debug then
5890         print()
5891         print((mode == 0) and '@@@@<' or '@@@@>', w)
5892       end
5893
5894       if nw == nil and w == '' then break end
5895
5896       if not lang then goto next end
5897       if not lbkr[lang] then goto next end
5898
5899       &% For each saved (pre|post)hyphenation. TODO. Reconsider how
5900       &% loops are nested.
5901       for k=1, #lbkr[lang] do
5902         local p = lbkr[lang][k].pattern
5903         local r = lbkr[lang][k].replace
5904
5905         if Babel.debug then
5906           print('*****', p, mode)
5907         end
5908
5909         &% This variable is set in some cases below to the first *byte*
5910         &% after the match, either as found by u.match (faster) or the
5911         &% computed position based on sc if w has changed.
5912         local last_match = 0
5913         local step = 0
5914
5915         &% For every match.
5916         while true do
5917           if Babel.debug then
5918             print('=====')
5919           end
5920           local new  &% used when inserting and removing nodes
5921
5922           local matches = { u.match(w, p, last_match) }
5923
5924           if #matches < 2 then break end
5925
5926           &% Get and remove empty captures (with ()'s, which return a
5927           &% number with the position), and keep actual captures
5928           &% (from (...)), if any, in matches.
5929           local first = table.remove(matches, 1)
5930           local last  = table.remove(matches, #matches)
5931           &% Non re-fetched substrings may contain \31, which separates
5932           &% subsubstrings.
5933           if string.find(w:sub(first, last-1), Babel.us_char) then break end
5934
5935           local save_last = last &% with A()BC()D, points to D
5936
5937           &% Fix offsets, from bytes to unicode. Explained above.
5938           first = u.len(w:sub(1, first-1)) + 1
5939           last  = u.len(w:sub(1, last-1)) &% now last points to C
5940
```

```
5941          &% This loop stores in n small table the nodes
5942          &% corresponding to the pattern. Used by 'data' to provide a
5943          &% predictable behavior with 'insert' (now w_nodes is modified on
5944          &% the fly), and also access to 'remove'd nodes.
5945          local sc = first-1          &% Used below, too
5946          local data_nodes = {}
5947
5948          for q = 1, last-first+1 do
5949            data_nodes[q] = w_nodes[sc+q]
5950          end
5951
5952          &% This loop traverses the matched substring and takes the
5953          &% corresponding action stored in the replacement list.
5954          &% sc = the position in substr nodes / string
5955          &% rc = the replacement table index
5956          local rc = 0
5957
5958          while rc < last-first+1 do &% for each replacement
5959            if Babel.debug then
5960              print('.....', rc + 1)
5961            end
5962            sc = sc + 1
5963            rc = rc + 1
5964
5965            if Babel.debug then
5966              Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
5967              local ss = ''
5968              for itt in node.traverse(head) do
5969               if itt.id == 29 then
5970                 ss = ss .. unicode.utf8.char(itt.char)
5971               else
5972                 ss = ss .. '{' .. itt.id .. '}'
5973               end
5974              end
5975              print('****************', ss)
5976
5977            end
5978
5979            local crep = r[rc]
5980            local item = w_nodes[sc]
5981            local item_base = item
5982            local placeholder = Babel.us_char
5983            local d
5984
5985            if crep and crep.data then
5986              item_base = data_nodes[crep.data]
5987            end
5988
5989            if crep then
5990              step = crep.step or 0
5991            end
5992
5993            if crep and next(crep) == nil then &% = {}
5994              last_match = save_last    &% Optimization
5995              goto next
5996
5997            elseif crep == nil or crep.remove then
5998              node.remove(head, item)
5999              table.remove(w_nodes, sc)
```

```
6000                  w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
6001                  sc = sc - 1  &% Nothing has been inserted.
6002                  last_match = utf8.offset(w, sc+1+step)
6003                  goto next
6004
6005             elseif crep and crep.kashida then &% Experimental
6006                node.set_attribute(item,
6007                   luatexbase.registernumber'bblar@kashida',
6008                   crep.kashida)
6009                last_match = utf8.offset(w, sc+1+step)
6010                goto next
6011
6012             elseif crep and crep.string then
6013                local str = crep.string(matches)
6014                if str == '' then  &% Gather with nil
6015                  node.remove(head, item)
6016                  table.remove(w_nodes, sc)
6017                  w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
6018                  sc = sc - 1  &% Nothing has been inserted.
6019                else
6020                  local loop_first = true
6021                  for s in string.utfvalues(str) do
6022                    d = node.copy(item_base)
6023                    d.char = s
6024                    if loop_first then
6025                      loop_first = false
6026                      head, new = node.insert_before(head, item, d)
6027                      if sc == 1 then
6028                        word_head = head
6029                      end
6030                      w_nodes[sc] = d
6031                      w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc+1)
6032                    else
6033                      sc = sc + 1
6034                      head, new = node.insert_before(head, item, d)
6035                      table.insert(w_nodes, sc, new)
6036                      w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc)
6037                    end
6038                    if Babel.debug then
6039                      print('.....', 'str')
6040                      Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6041                    end
6042                  end  &% for
6043                  node.remove(head, item)
6044                end  &% if ''
6045                last_match = utf8.offset(w, sc+1+step)
6046                goto next
6047
6048             elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
6049                d = node.new(7, 0)   &% (disc, discretionary)
6050                d.pre     = Babel.str_to_nodes(crep.pre, matches, item_base)
6051                d.post    = Babel.str_to_nodes(crep.post, matches, item_base)
6052                d.replace = Babel.str_to_nodes(crep.no, matches, item_base)
6053                d.attr = item_base.attr
6054                if crep.pre == nil then  &% TeXbook p96
6055                  d.penalty = crep.penalty or tex.hyphenpenalty
6056                else
6057                  d.penalty = crep.penalty or tex.exhyphenpenalty
6058                end
```

```
6059            placeholder = '|'
6060            head, new = node.insert_before(head, item, d)
6061

6062          elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
6063            &% ERROR
6064

6065          elseif crep and crep.penalty then
6066            d = node.new(14, 0)   &% (penalty, userpenalty)
6067            d.attr = item_base.attr
6068            d.penalty = crep.penalty
6069            head, new = node.insert_before(head, item, d)
6070

6071          elseif crep and crep.space then
6072            &% 655360 = 10 pt = 10 * 65536 sp
6073            d = node.new(12, 13)      &% (glue, spaceskip)
6074            local quad = font.getfont(item_base.font).size or 655360
6075            node.setglue(d, crep.space[1] * quad,
6076                            crep.space[2] * quad,
6077                            crep.space[3] * quad)
6078            if mode == 0 then
6079              placeholder = ' '
6080            end
6081            head, new = node.insert_before(head, item, d)
6082

6083          elseif crep and crep.spacefactor then
6084            d = node.new(12, 13)      &% (glue, spaceskip)
6085            local base_font = font.getfont(item_base.font)
6086            node.setglue(d,
6087              crep.spacefactor[1] * base_font.parameters['space'],
6088              crep.spacefactor[2] * base_font.parameters['space_stretch'],
6089              crep.spacefactor[3] * base_font.parameters['space_shrink'])
6090            if mode == 0 then
6091              placeholder = ' '
6092            end
6093            head, new = node.insert_before(head, item, d)
6094

6095          elseif mode == 0 and crep and crep.space then
6096            &% ERROR
6097

6098          end  &% ie replacement cases
6099

6100          &% Shared by disc, space and penalty.
6101          if sc == 1 then
6102            word_head = head
6103          end
6104          if crep.insert then
6105            w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc)
6106            table.insert(w_nodes, sc, new)
6107            last = last + 1
6108          else
6109            w_nodes[sc] = d
6110            node.remove(head, item)
6111            w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1)
6112          end
6113

6114          last_match = utf8.offset(w, sc+1+step)
6115

6116          ::next::
6117
```

```
6118            end   &% for each replacement
6119
6120           if Babel.debug then
6121               print('.....', '/')
6122               Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6123           end
6124
6125         end   &% for match
6126
6127      end   &% for patterns
6128
6129      ::next::
6130      word_head = nw
6131    end   &% for substring
6132    return head
6133  end
6134
6135  &% This table stores capture maps, numbered consecutively
6136  Babel.capture_maps = {}
6137
6138  &% The following functions belong to the next macro
6139  function Babel.capture_func(key, cap)
6140    local ret = "[[" .. cap:gsub('{([0-9])}', "]]..m[%1]..[[") .. "]]"
6141    local cnt
6142    local u = unicode.utf8
6143    ret, cnt = ret:gsub('{([0-9])|([^|]+)|(.-)}', Babel.capture_func_map)
6144    if cnt == 0 then
6145      ret = u.gsub(ret, '{(%x%x%x%x+)}',
6146            function (n)
6147              return u.char(tonumber(n, 16))
6148            end)
6149    end
6150    ret = ret:gsub("%[%[%]%]%.%.", '')
6151    ret = ret:gsub("%.%.%[%[%]%]", '')
6152    return key .. [[=function(m) return ]] .. ret .. [[ end]]
6153  end
6154
6155  function Babel.capt_map(from, mapno)
6156    return Babel.capture_maps[mapno][from] or from
6157  end
6158
6159  &% Handle the {n|abc|ABC} syntax in captures
6160  function Babel.capture_func_map(capno, from, to)
6161    local u = unicode.utf8
6162    from = u.gsub(from, '{(%x%x%x%x+)}',
6163          function (n)
6164            return u.char(tonumber(n, 16))
6165          end)
6166    to = u.gsub(to, '{(%x%x%x%x+)}',
6167          function (n)
6168            return u.char(tonumber(n, 16))
6169          end)
6170    local froms = {}
6171    for s in string.utfcharacters(from) do
6172      table.insert(froms, s)
6173    end
6174    local cnt = 1
6175    table.insert(Babel.capture_maps, {})
6176    local mlen = table.getn(Babel.capture_maps)
```

```
6177       for s in string.utfcharacters(to) do
6178         Babel.capture_maps[mlen][froms[cnt]] = s
6179         cnt = cnt + 1
6180       end
6181       return "]]..Babel.capt_map(m[" .. capno .. "]," ..
6182              (mlen) .. ")..".. "[["
6183     end
6184
6185     &% Create/Extend reversed sorted list of kashida weights:
6186     function Babel.capture_kashida(key, wt)
6187       wt = tonumber(wt)
6188       if Babel.kashida_wts then
6189         for p, q in ipairs(Babel.kashida_wts) do
6190           if wt  == q then
6191             break
6192           elseif wt > q then
6193             table.insert(Babel.kashida_wts, p, wt)
6194             break
6195           elseif table.getn(Babel.kashida_wts) == p then
6196             table.insert(Babel.kashida_wts, wt)
6197           end
6198         end
6199       else
6200         Babel.kashida_wts = { wt }
6201       end
6202       return 'kashida = ' .. wt
6203     end
6204 }
```

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).

```
6205 \catcode`\#=6
6206 \gdef\babelposthyphenation#1#2#3{&%
6207   \bbl@activateposthyphen
6208   \begingroup
6209     \def\babeltempa{\bbl@add@list\babeltempb}&%
6210     \let\babeltempb\@empty
6211     \def\bbl@tempa{#3}&% TODO. Ugly trick to preserve {}:
6212     \bbl@replace\bbl@tempa{,}{ ,}&%
6213     \expandafter\bbl@foreach\expandafter{\bbl@tempa}{&%
6214       \bbl@ifsamestring{##1}{remove}&%
6215         {\bbl@add@list\babeltempb{nil}}&%
6216         {\directlua{
6217            local rep = [=[##1]=]
6218            rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
6219            rep = rep:gsub('^%s*(insert)%s*,', 'insert = true, ')
6220            rep = rep:gsub(    '(no)%s*=%s*([^%s,]*)', Babel.capture_func)
6221            rep = rep:gsub(   '(pre)%s*=%s*([^%s,]*)', Babel.capture_func)
6222            rep = rep:gsub(  '(post)%s*=%s*([^%s,]*)', Babel.capture_func)
6223            rep = rep:gsub('(string)%s*=%s*([^%s,]*)', Babel.capture_func)
6224            tex.print([[\string\babeltempa{{]] .. rep .. [[}}]])
6225         }}}&%
```

```
6226    \directlua{
6227      local lbkr = Babel.linebreaking.replacements[1]
6228      local u = unicode.utf8
6229      local id = \the\csname l@#1\endcsname
6230      &% Convert pattern:
6231      local patt = string.gsub([==[#2]==], '%s', '')
6232      if not u.find(patt, '()', nil, true) then
6233        patt = '()' .. patt .. '()'
6234      end
6235      patt = string.gsub(patt, '%(%)%^', '^()')
6236      patt = string.gsub(patt, '%$%(%)', '()$')
6237      patt = u.gsub(patt, '{(.)}',
6238            function (n)
6239              return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
6240            end)
6241      patt = u.gsub(patt, '{(%x%x%x%x+)}',
6242            function (n)
6243              return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%%1')
6244            end)
6245      lbkr[id] = lbkr[id] or {}
6246      table.insert(lbkr[id], { pattern = patt, replace = { \babeltempb } })
6247    }&%
6248  \endgroup}
6249 % TODO. Copypaste pattern.
6250 \gdef\babelprehyphenation#1#2#3{&%
6251   \bbl@activateprehyphen
6252   \begingroup
6253     \def\babeltempa{\bbl@add@list\babeltempb}&%
6254     \let\babeltempb\@empty
6255     \def\bbl@tempa{#3}&% TODO. Ugly trick to preserve {}:
6256     \bbl@replace\bbl@tempa{,}{ ,}&%
6257     \expandafter\bbl@foreach\expandafter{\bbl@tempa}{&%
6258       \bbl@ifsamestring{##1}{remove}&%
6259         {\bbl@add@list\babeltempb{nil}}&%
6260         {\directlua{
6261            local rep = [=[##1]=]
6262            rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
6263            rep = rep:gsub('^%s*(insert)%s*,', 'insert = true, ')
6264            rep = rep:gsub('(string)%s*=%s*([^%s,]*)', Babel.capture_func)
6265            rep = rep:gsub('(space)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
6266              'space = {' .. '%2, %3, %4' .. '}')
6267            rep = rep:gsub('(spacefactor)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
6268              'spacefactor = {' .. '%2, %3, %4' .. '}')
6269            rep = rep:gsub('(kashida)%s*=%s*([^%s,]*)', Babel.capture_kashida)
6270            tex.print([[\string\babeltempa{{] .. rep .. [[}}]])
6271         }}}&%
6272     \directlua{
6273       local lbkr = Babel.linebreaking.replacements[0]
6274       local u = unicode.utf8
6275       local id = \the\csname bbl@id@@#1\endcsname
6276       &% Convert pattern:
6277       local patt = string.gsub([==[#2]==], '%s', '')
6278       local patt = string.gsub(patt, '|', ' ')
6279       if not u.find(patt, '()', nil, true) then
6280         patt = '()' .. patt .. '()'
6281       end
6282       &% patt = string.gsub(patt, '%(%)%^', '^()')
6283       &% patt = string.gsub(patt, '([^%%])%$%(%)', '%1()$')
6284       patt = u.gsub(patt, '{(.)}',
```

```
6285            function (n)
6286              return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
6287            end)
6288        patt = u.gsub(patt, '{(%x%x%x%x+)}',
6289                function (n)
6290                  return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%%1')
6291                end)
6292        lbkr[id] = lbkr[id] or {}
6293        table.insert(lbkr[id], { pattern = patt, replace = { \babeltempb } })
6294      }&%
6295    \endgroup}
6296  \endgroup
6297  \def\bbl@activateposthyphen{%
6298    \let\bbl@activateposthyphen\relax
6299    \directlua{
6300      Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
6301    }}
6302  \def\bbl@activateprehyphen{%
6303    \let\bbl@activateprehyphen\relax
6304    \directlua{
6305      Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
6306    }}
```

13.9   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.

```
6307  \bbl@trace{Redefinitions for bidi layout}
6308  \ifx\@eqnnum\@undefined\else
6309    \ifx\bbl@attr@dir\@undefined\else
6310      \edef\@eqnnum{{%
6311        \unexpanded{\ifcase\bbl@attr@dir\else\bbl@textdir\@ne\fi}%
6312        \unexpanded\expandafter{\@eqnnum}}}
6313    \fi
6314  \fi
6315  \ifx\bbl@opt@layout\@nnil\endinput\fi  % if no layout
6316  \ifnum\bbl@bidimode>\z@
6317    \def\bbl@nextfake#1{%  non-local changes, use always inside a group!
6318      \bbl@exp{%
6319        \mathdir\the\bodydir
6320        #1%                Once entered in math, set boxes to restore values
6321        \<ifmmode>%
6322          \everyvbox{%
6323            \the\everyvbox
6324            \bodydir\the\bodydir
6325            \mathdir\the\mathdir
6326            \everyhbox{\the\everyhbox}%
6327            \everyvbox{\the\everyvbox}}%
```

```
6328        \everyhbox{%
6329          \the\everyhbox
6330          \bodydir\the\bodydir
6331          \mathdir\the\mathdir
6332          \everyhbox{\the\everyhbox}%
6333          \everyvbox{\the\everyvbox}}%
6334      \<fi>}}%
6335    \def\@hangfrom#1{%
6336      \setbox\@tempboxa\hbox{{#1}}%
6337      \hangindent\wd\@tempboxa
6338      \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
6339        \shapemode\@ne
6340      \fi
6341      \noindent\box\@tempboxa}
6342  \fi
6343  \IfBabelLayout{tabular}
6344    {\let\bbl@OL@@tabular\@tabular
6345     \bbl@replace\@tabular{$}{\bbl@nextfake$}%
6346     \let\bbl@NL@@tabular\@tabular
6347     \AtBeginDocument{%
6348       \ifx\bbl@NL@@tabular\@tabular\else
6349         \bbl@replace\@tabular{$}{\bbl@nextfake$}%
6350         \let\bbl@NL@@tabular\@tabular
6351       \fi}}
6352    {}
6353  \IfBabelLayout{lists}
6354    {\let\bbl@OL@list\list
6355     \bbl@sreplace\list{\parshape}{\bbl@listparshape}%
6356     \let\bbl@NL@list\list
6357     \def\bbl@listparshape#1#2#3{%
6358       \parshape #1 #2 #3 %
6359       \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
6360         \shapemode\tw@
6361       \fi}}
6362    {}
6363  \IfBabelLayout{graphics}
6364    {\let\bbl@pictresetdir\relax
6365     \def\bbl@pictsetdir#1{%
6366       \ifcase\bbl@thetextdir
6367         \let\bbl@pictresetdir\relax
6368       \else
6369         \ifcase#1\bodydir TLT  % Remember this sets the inner boxes
6370           \or\textdir TLT
6371           \else\bodydir TLT \textdir TLT
6372         \fi
6373         % \(text|par)dir required in pgf:
6374         \def\bbl@pictresetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}%
6375       \fi}%
6376     \ifx\AddToHook\@undefined\else
6377       \AddToHook{env/picture/begin}{\bbl@pictsetdir\tw@}%
6378       \directlua{
6379         Babel.get_picture_dir = true
6380         Babel.picture_has_bidi = 0
6381         function Babel.picture_dir (head)
6382           if not Babel.get_picture_dir then return head end
6383           for item in node.traverse(head) do
6384             if item.id == node.id'glyph' then
6385               local itemchar = item.char
6386               % TODO. Copypaste pattern from Babel.bidi (-r)
```

197

```
6387              local chardata = Babel.characters[itemchar]
6388              local dir = chardata and chardata.d or nil
6389              if not dir then
6390                for nn, et in ipairs(Babel.ranges) do
6391                  if itemchar < et[1] then
6392                    break
6393                  elseif itemchar <= et[2] then
6394                    dir = et[3]
6395                    break
6396                  end
6397                end
6398              end
6399              if dir and (dir == 'al' or dir == 'r') then
6400                Babel.picture_has_bidi = 1
6401              end
6402            end
6403          end
6404          return head
6405        end
6406        luatexbase.add_to_callback("hpack_filter", Babel.picture_dir,
6407          "Babel.picture_dir")
6408      }%
6409    \AtBeginDocument{%
6410      \long\def\put(#1,#2)#3{%
6411        \@killglue
6412        % Try:
6413        \ifx\bbl@pictresetdir\relax
6414          \def\bbl@tempc{0}%
6415        \else
6416          \directlua{
6417            Babel.get_picture_dir = true
6418            Babel.picture_has_bidi = 0
6419          }%
6420          \setbox\z@\hb@xt@\z@{%
6421            \@defaultunitsset\@tempdimc{#1}\unitlength
6422            \kern\@tempdimc
6423            #3\hss}%
6424          \edef\bbl@tempc{\directlua{tex.print(Babel.picture_has_bidi)}}%
6425        \fi
6426        % Do:
6427        \@defaultunitsset\@tempdimc{#2}\unitlength
6428        \raise\@tempdimc\hb@xt@\z@{%
6429          \@defaultunitsset\@tempdimc{#1}\unitlength
6430          \kern\@tempdimc
6431          {\ifnum\bbl@tempc>\z@\bbl@pictresetdir\fi#3}\hss}%
6432        \ignorespaces}%
6433      \MakeRobust\put}%
6434    \fi
6435    \AtBeginDocument
6436      {\ifx\tikz@atbegin@node\@undefined\else
6437        \ifx\AddToHook\@undefined\else % TODO. Still tentative.
6438          \AddToHook{env/pgfpicture/begin}{\bbl@pictsetdir\@ne}%
6439          \bbl@add\pgfinterruptpicture{\bbl@pictresetdir}%
6440        \fi
6441        \let\bbl@OL@pgfpicture\pgfpicture
6442        \bbl@sreplace\pgfpicture{\pgfpicturetrue}%
6443          {\bbl@pictsetdir\z@\pgfpicturetrue}%
6444        \bbl@add\pgfsys@beginpicture{\bbl@pictsetdir\z@}%
6445        \bbl@add\tikz@atbegin@node{\bbl@pictresetdir}%
```

```
6446        \bbl@sreplace\tikz{\begingroup}%
6447          {\begingroup\bbl@pictsetdir\tw@}%
6448      \fi
6449      \ifx\AddToHook\@undefined\else
6450        \AddToHook{env/tcolorbox/begin}{\bbl@pictsetdir\@ne}%
6451      \fi
6452      }}
6453    {}
```

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`.

```
6454 \IfBabelLayout{counters}%
6455    {\let\bbl@OL@@textsuperscript\@textsuperscript
6456      \bbl@sreplace\@textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
6457      \let\bbl@latinarabic=\@arabic
6458      \let\bbl@OL@@arabic\@arabic
6459      \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
6460      \@ifpackagewith{babel}{bidi=default}%
6461        {\let\bbl@asciiroman=\@roman
6462        \let\bbl@OL@@roman\@roman
6463        \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciiroman#1}}}%
6464        \let\bbl@asciiRoman=\@Roman
6465        \let\bbl@OL@@roman\@Roman
6466        \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}%
6467        \let\bbl@OL@labelenumii\labelenumii
6468        \def\labelenumii{)\theenumii(}%
6469        \let\bbl@OL@p@enumiii\p@enumiii
6470        \def\p@enumiii{\p@enumii)\theenumii(}}{}}{}}
6471 ⟨⟨Footnote changes⟩⟩
6472 \IfBabelLayout{footnotes}%
6473    {\let\bbl@OL@footnote\footnote
6474      \BabelFootnote\footnote\languagename{}{}%
6475      \BabelFootnote\localfootnote\languagename{}{}%
6476      \BabelFootnote\mainfootnote{}{}{}}
6477    {}
```

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.

```
6478 \IfBabelLayout{extras}%
6479    {\let\bbl@OL@underline\underline
6480      \bbl@sreplace\underline{$\@@underline}{\bbl@nextfake$\@@underline}%
6481      \let\bbl@OL@LaTeX2e\LaTeX2e
6482      \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
6483        \if b\expandafter\@car\f@series\@nil\boldmath\fi
6484        \babelsublr{%
6485          \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}}}
6486    {}
6487 ⟨/luatex⟩
```

13.10   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.

```
6488 ⟨*basic-r⟩
6489 Babel = Babel or {}
6490
6491 Babel.bidi_enabled = true
6492
6493 require('babel-data-bidi.lua')
6494
6495 local characters = Babel.characters
6496 local ranges = Babel.ranges
6497
6498 local DIR = node.id("dir")
6499
6500 local function dir_mark(head, from, to, outer)
6501   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
6502   local d = node.new(DIR)
6503   d.dir = '+' .. dir
6504   node.insert_before(head, from, d)
6505   d = node.new(DIR)
6506   d.dir = '-' .. dir
6507   node.insert_after(head, to, d)
6508 end
6509
6510 function Babel.bidi(head, ispar)
6511   local first_n, last_n          -- first and last char with nums
6512   local last_es                  -- an auxiliary 'last' used with nums
6513   local first_d, last_d          -- first and last char in L/R block
6514   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):

```
6515  local strong = ('TRT' == tex.pardir) and 'r' or 'l'
6516  local strong_lr = (strong == 'l') and 'l' or 'r'
6517  local outer = strong
6518
6519  local new_dir = false
6520  local first_dir = false
6521  local inmath = false
6522
6523  local last_lr
6524
6525  local type_n = ''
6526
6527  for item in node.traverse(head) do
6528
6529    -- three cases: glyph, dir, otherwise
6530    if item.id == node.id'glyph'
6531      or (item.id == 7 and item.subtype == 2) then
6532
6533      local itemchar
6534      if item.id == 7 and item.subtype == 2 then
6535        itemchar = item.replace.char
6536      else
6537        itemchar = item.char
6538      end
6539      local chardata = characters[itemchar]
6540      dir = chardata and chardata.d or nil
6541      if not dir then
6542        for nn, et in ipairs(ranges) do
6543          if itemchar < et[1] then
6544            break
6545          elseif itemchar <= et[2] then
6546            dir = et[3]
6547            break
6548          end
6549        end
6550      end
6551      dir = dir or 'l'
6552      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).

```
6553      if new_dir then
6554        attr_dir = 0
6555        for at in node.traverse(item.attr) do
6556          if at.number == luatexbase.registernumber'bbl@attr@dir' then
6557            attr_dir = at.value % 3
6558          end
6559        end
6560        if attr_dir == 1 then
6561          strong = 'r'
6562        elseif attr_dir == 2 then
6563          strong = 'al'
6564        else
```

```
6565          strong = 'l'
6566        end
6567        strong_lr = (strong == 'l') and 'l' or 'r'
6568        outer = strong_lr
6569        new_dir = false
6570      end
6571
6572      if dir == 'nsm' then dir = strong end            -- W1
```

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

```
6573      dir_real = dir                -- We need dir_real to set strong below
6574      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:

```
6575      if strong == 'al' then
6576        if dir == 'en' then dir = 'an' end                 -- W2
6577        if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
6578        strong_lr = 'r'                                    -- W3
6579      end
```

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

```
6580    elseif item.id == node.id'dir' and not inmath then
6581      new_dir = true
6582      dir = nil
6583    elseif item.id == node.id'math' then
6584      inmath = (item.subtype == 0)
6585    else
6586      dir = nil          -- Not a char
6587    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>.

```
6588    if dir == 'en' or dir == 'an' or dir == 'et' then
6589      if dir ~= 'et' then
6590        type_n = dir
6591      end
6592      first_n = first_n or item
6593      last_n = last_es or item
6594      last_es = nil
6595    elseif dir == 'es' and last_n then -- W3+W6
6596      last_es = item
6597    elseif dir == 'cs' then          -- it's right - do nothing
6598    elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
6599      if strong_lr == 'r' and type_n ~= '' then
6600        dir_mark(head, first_n, last_n, 'r')
6601      elseif strong_lr == 'l' and first_d and type_n == 'an' then
6602        dir_mark(head, first_n, last_n, 'r')
6603        dir_mark(head, first_d, last_d, outer)
6604        first_d, last_d = nil, nil
6605      elseif strong_lr == 'l' and type_n ~= '' then
6606        last_d = last_n
6607      end
6608      type_n = ''
6609      first_n, last_n = nil, nil
6610    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:

```
6611    if dir == 'l' or dir == 'r' then
6612      if dir ~= outer then
6613        first_d = first_d or item
6614        last_d = item
6615      elseif first_d and dir ~= strong_lr then
6616        dir_mark(head, first_d, last_d, outer)
6617        first_d, last_d = nil, nil
6618      end
6619    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.

```
6620    if dir and not last_lr and dir ~= 'l' and outer == 'r' then
6621      item.char = characters[item.char] and
6622                  characters[item.char].m or item.char
6623    elseif (dir or new_dir) and last_lr ~= item then
6624      local mir = outer .. strong_lr .. (dir or outer)
6625      if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
6626        for ch in node.traverse(node.next(last_lr)) do
6627          if ch == item then break end
6628          if ch.id == node.id'glyph' and characters[ch.char] then
6629            ch.char = characters[ch.char].m or ch.char
6630          end
6631        end
6632      end
6633    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).

```
6634    if dir == 'l' or dir == 'r' then
6635      last_lr = item
6636      strong = dir_real              -- Don't search back - best save now
6637      strong_lr = (strong == 'l') and 'l' or 'r'
6638    elseif new_dir then
6639      last_lr = nil
6640    end
6641  end
```

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

```
6642  if last_lr and outer == 'r' then
6643    for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
6644      if characters[ch.char] then
6645        ch.char = characters[ch.char].m or ch.char
6646      end
6647    end
6648  end
6649  if first_n then
6650    dir_mark(head, first_n, last_n, outer)
6651  end
6652  if first_d then
6653    dir_mark(head, first_d, last_d, outer)
6654  end
```

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

```
6655   return node.prev(head) or head
6656 end
6657 ⟨/basic-r⟩
```

And here the Lua code for bidi=basic:

```
6658 ⟨∗basic⟩
6659 Babel = Babel or {}
6660
6661 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
6662
6663 Babel.fontmap = Babel.fontmap or {}
6664 Babel.fontmap[0] = {}        -- l
6665 Babel.fontmap[1] = {}        -- r
6666 Babel.fontmap[2] = {}        -- al/an
6667
6668 Babel.bidi_enabled = true
6669 Babel.mirroring_enabled = true
6670
6671 require('babel-data-bidi.lua')
6672
6673 local characters = Babel.characters
6674 local ranges = Babel.ranges
6675
6676 local DIR = node.id('dir')
6677 local GLYPH = node.id('glyph')
6678
6679 local function insert_implicit(head, state, outer)
6680   local new_state = state
6681   if state.sim and state.eim and state.sim ~= state.eim then
6682     dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
6683     local d = node.new(DIR)
6684     d.dir = '+' .. dir
6685     node.insert_before(head, state.sim, d)
6686     local d = node.new(DIR)
6687     d.dir = '-' .. dir
6688     node.insert_after(head, state.eim, d)
6689   end
6690   new_state.sim, new_state.eim = nil, nil
6691   return head, new_state
6692 end
6693
6694 local function insert_numeric(head, state)
6695   local new
6696   local new_state = state
6697   if state.san and state.ean and state.san ~= state.ean then
6698     local d = node.new(DIR)
6699     d.dir = '+TLT'
6700     _, new = node.insert_before(head, state.san, d)
6701     if state.san == state.sim then state.sim = new end
6702     local d = node.new(DIR)
6703     d.dir = '-TLT'
6704     _, new = node.insert_after(head, state.ean, d)
6705     if state.ean == state.eim then state.eim = new end
6706   end
6707   new_state.san, new_state.ean = nil, nil
6708   return head, new_state
6709 end
```

```lua
6710
6711 -- TODO - \hbox with an explicit dir can lead to wrong results
6712 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
6713 -- was s made to improve the situation, but the problem is the 3-dir
6714 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
6715 -- well.
6716
6717 function Babel.bidi(head, ispar, hdir)
6718   local d    -- d is used mainly for computations in a loop
6719   local prev_d = ''
6720   local new_d = false
6721
6722   local nodes = {}
6723   local outer_first = nil
6724   local inmath = false
6725
6726   local glue_d = nil
6727   local glue_i = nil
6728
6729   local has_en = false
6730   local first_et = nil
6731
6732   local ATDIR = luatexbase.registernumber'bbl@attr@dir'
6733
6734   local save_outer
6735   local temp = node.get_attribute(head, ATDIR)
6736   if temp then
6737     temp = temp % 3
6738     save_outer = (temp == 0 and 'l') or
6739                  (temp == 1 and 'r') or
6740                  (temp == 2 and 'al')
6741   elseif ispar then            -- Or error? Shouldn't happen
6742     save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
6743   else                         -- Or error? Shouldn't happen
6744     save_outer = ('TRT' == hdir) and 'r' or 'l'
6745   end
6746     -- when the callback is called, we are just _after_ the box,
6747     -- and the textdir is that of the surrounding text
6748   -- if not ispar and hdir ~= tex.textdir then
6749   --   save_outer = ('TRT' == hdir) and 'r' or 'l'
6750   -- end
6751   local outer = save_outer
6752   local last = outer
6753   -- 'al' is only taken into account in the first, current loop
6754   if save_outer == 'al' then save_outer = 'r' end
6755
6756   local fontmap = Babel.fontmap
6757
6758   for item in node.traverse(head) do
6759
6760     -- In what follows, #node is the last (previous) node, because the
6761     -- current one is not added until we start processing the neutrals.
6762
6763     -- three cases: glyph, dir, otherwise
6764     if item.id == GLYPH
6765        or (item.id == 7 and item.subtype == 2) then
6766
6767       local d_font = nil
6768       local item_r
```

```
6769        if item.id == 7 and item.subtype == 2 then
6770          item_r = item.replace    -- automatic discs have just 1 glyph
6771        else
6772          item_r = item
6773        end
6774        local chardata = characters[item_r.char]
6775        d = chardata and chardata.d or nil
6776        if not d or d == 'nsm' then
6777          for nn, et in ipairs(ranges) do
6778            if item_r.char < et[1] then
6779              break
6780            elseif item_r.char <= et[2] then
6781              if not d then d = et[3]
6782              elseif d == 'nsm' then d_font = et[3]
6783              end
6784              break
6785            end
6786          end
6787        end
6788        d = d or 'l'
6789
6790        -- A short 'pause' in bidi for mapfont
6791        d_font = d_font or d
6792        d_font = (d_font == 'l' and 0) or
6793                 (d_font == 'nsm' and 0) or
6794                 (d_font == 'r' and 1) or
6795                 (d_font == 'al' and 2) or
6796                 (d_font == 'an' and 2) or nil
6797        if d_font and fontmap and fontmap[d_font][item_r.font] then
6798          item_r.font = fontmap[d_font][item_r.font]
6799        end
6800
6801        if new_d then
6802          table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
6803          if inmath then
6804            attr_d = 0
6805          else
6806            attr_d = node.get_attribute(item, ATDIR)
6807            attr_d = attr_d % 3
6808          end
6809          if attr_d == 1 then
6810            outer_first = 'r'
6811            last = 'r'
6812          elseif attr_d == 2 then
6813            outer_first = 'r'
6814            last = 'al'
6815          else
6816            outer_first = 'l'
6817            last = 'l'
6818          end
6819          outer = last
6820          has_en = false
6821          first_et = nil
6822          new_d = false
6823        end
6824
6825        if glue_d then
6826          if (d == 'l' and 'l' or 'r') ~= glue_d then
6827              table.insert(nodes, {glue_i, 'on', nil})
```

```
6828          end
6829          glue_d = nil
6830          glue_i = nil
6831        end
6832
6833      elseif item.id == DIR then
6834        d = nil
6835        new_d = true
6836
6837      elseif item.id == node.id'glue' and item.subtype == 13 then
6838        glue_d = d
6839        glue_i = item
6840        d = nil
6841
6842      elseif item.id == node.id'math' then
6843        inmath = (item.subtype == 0)
6844
6845      else
6846        d = nil
6847      end
6848
6849      -- AL <= EN/ET/ES      -- W2 + W3 + W6
6850      if last == 'al' and d == 'en' then
6851        d = 'an'            -- W3
6852      elseif last == 'al' and (d == 'et' or d == 'es') then
6853        d = 'on'            -- W6
6854      end
6855
6856      -- EN + CS/ES + EN      -- W4
6857      if d == 'en' and #nodes >= 2 then
6858        if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
6859            and nodes[#nodes-1][2] == 'en' then
6860          nodes[#nodes][2] = 'en'
6861        end
6862      end
6863
6864      -- AN + CS + AN         -- W4 too, because uax9 mixes both cases
6865      if d == 'an' and #nodes >= 2 then
6866        if (nodes[#nodes][2] == 'cs')
6867            and nodes[#nodes-1][2] == 'an' then
6868          nodes[#nodes][2] = 'an'
6869        end
6870      end
6871
6872      -- ET/EN                -- W5 + W7->l / W6->on
6873      if d == 'et' then
6874        first_et = first_et or (#nodes + 1)
6875      elseif d == 'en' then
6876        has_en = true
6877        first_et = first_et or (#nodes + 1)
6878      elseif first_et then        -- d may be nil here !
6879        if has_en then
6880          if last == 'l' then
6881            temp = 'l'     -- W7
6882          else
6883            temp = 'en'    -- W5
6884          end
6885        else
6886          temp = 'on'      -- W6
```

```
6887        end
6888        for e = first_et, #nodes do
6889          if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
6890        end
6891        first_et = nil
6892        has_en = false
6893      end
6894
6895      -- Force mathdir in math if ON (currently works as expected only
6896      -- with 'l')
6897      if inmath and d == 'on' then
6898        d = ('TRT' == tex.mathdir) and 'r' or 'l'
6899      end
6900
6901      if d then
6902        if d == 'al' then
6903          d = 'r'
6904          last = 'al'
6905        elseif d == 'l' or d == 'r' then
6906          last = d
6907        end
6908        prev_d = d
6909        table.insert(nodes, {item, d, outer_first})
6910      end
6911
6912    outer_first = nil
6913
6914  end
6915
6916  -- TODO -- repeated here in case EN/ET is the last node. Find a
6917  -- better way of doing things:
6918  if first_et then       -- dir may be nil here !
6919    if has_en then
6920      if last == 'l' then
6921        temp = 'l'    -- W7
6922      else
6923        temp = 'en'   -- W5
6924      end
6925    else
6926      temp = 'on'     -- W6
6927    end
6928    for e = first_et, #nodes do
6929      if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
6930    end
6931  end
6932
6933  -- dummy node, to close things
6934  table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
6935
6936  --------------  NEUTRAL -----------------
6937
6938  outer = save_outer
6939  last = outer
6940
6941  local first_on = nil
6942
6943  for q = 1, #nodes do
6944    local item
6945
```

```
6946    local outer_first = nodes[q][3]
6947    outer = outer_first or outer
6948    last = outer_first or last
6949
6950    local d = nodes[q][2]
6951    if d == 'an' or d == 'en' then d = 'r' end
6952    if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
6953
6954    if d == 'on' then
6955      first_on = first_on or q
6956    elseif first_on then
6957      if last == d then
6958        temp = d
6959      else
6960        temp = outer
6961      end
6962      for r = first_on, q - 1 do
6963        nodes[r][2] = temp
6964        item = nodes[r][1]    -- MIRRORING
6965        if Babel.mirroring_enabled and item.id == GLYPH
6966            and temp == 'r' and characters[item.char] then
6967          local font_mode = font.fonts[item.font].properties.mode
6968          if font_mode ~= 'harf' and font_mode ~= 'plug' then
6969            item.char = characters[item.char].m or item.char
6970          end
6971        end
6972      end
6973      first_on = nil
6974    end
6975
6976    if d == 'r' or d == 'l' then last = d end
6977  end
6978
6979  -------------  IMPLICIT, REORDER ----------------
6980
6981  outer = save_outer
6982  last = outer
6983
6984  local state = {}
6985  state.has_r = false
6986
6987  for q = 1, #nodes do
6988
6989    local item = nodes[q][1]
6990
6991    outer = nodes[q][3] or outer
6992
6993    local d = nodes[q][2]
6994
6995    if d == 'nsm' then d = last end               -- W1
6996    if d == 'en' then d = 'an' end
6997    local isdir = (d == 'r' or d == 'l')
6998
6999    if outer == 'l' and d == 'an' then
7000      state.san = state.san or item
7001      state.ean = item
7002    elseif state.san then
7003      head, state = insert_numeric(head, state)
7004    end
```

```
7005
7006    if outer == 'l' then
7007      if d == 'an' or d == 'r' then      -- im -> implicit
7008        if d == 'r' then state.has_r = true end
7009        state.sim = state.sim or item
7010        state.eim = item
7011      elseif d == 'l' and state.sim and state.has_r then
7012        head, state = insert_implicit(head, state, outer)
7013      elseif d == 'l' then
7014        state.sim, state.eim, state.has_r = nil, nil, false
7015      end
7016    else
7017      if d == 'an' or d == 'l' then
7018        if nodes[q][3] then -- nil except after an explicit dir
7019          state.sim = item  -- so we move sim 'inside' the group
7020        else
7021          state.sim = state.sim or item
7022        end
7023        state.eim = item
7024      elseif d == 'r' and state.sim then
7025        head, state = insert_implicit(head, state, outer)
7026      elseif d == 'r' then
7027        state.sim, state.eim = nil, nil
7028      end
7029    end
7030
7031    if isdir then
7032      last = d           -- Don't search back - best save now
7033    elseif d == 'on' and state.san  then
7034      state.san = state.san or item
7035      state.ean = item
7036    end
7037
7038  end
7039
7040  return node.prev(head) or head
7041 end
7042 ⟨/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.

```
7043 ⟨∗nil⟩
7044 \ProvidesLanguage{nil}[⟨⟨date⟩⟩ ⟨⟨version⟩⟩ Nil language]
7045 \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.

```
7046 \ifx\l@nil\@undefined
7047   \newlanguage\l@nil
7048   \@namedef{bbl@hyphendata@\the\l@nil}{{}{}}% Remove warning
7049   \let\bbl@elt\relax
7050   \edef\bbl@languages{%  Add it to the list of languages
7051     \bbl@languages\bbl@elt{nil}{\the\l@nil}{}{}}
7052 \fi
```

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

```
7053 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}
```

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

`\captionnil`
`\datenil`
```
7054 \let\captionsnil\@empty
7055 \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.

```
7056 \ldf@finish{nil}
7057 ⟨/nil⟩
```

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

16.1   Not renaming `hyphen.tex`

As Don Knuth has declared that the filename `hyphen.tex` may only be used to designate *his* version of the american English hyphenation patterns, a new solution has to be found in order to be able to load hyphenation patterns for other languages in a plain-based TEX-format. When asked he responded:

> That file name is "sacred", and if anybody changes it they will cause severe upward/downward compatibility headaches.

> People can have a file localhyphen.tex or whatever they like, but they mustn't diddle with hyphen.tex (or plain.tex except to preload additional fonts).

The files `bplain.tex` and `blplain.tex` can be used as replacement wrappers around `plain.tex` and `lplain.tex` to achieve the desired effect, based on the babel package. If you load each of them with iniTEX, you will get a file called either `bplain.fmt` or `blplain.fmt`, which you can use as replacements for `plain.fmt` and `lplain.fmt`.
As these files are going to be read as the first thing iniTEX sees, we need to set some category codes just to be able to change the definition of `\input`.

```
7058 ⟨∗bplain | blplain⟩
7059 \catcode`\{=1 % left brace is begin-group character
7060 \catcode`\}=2 % right brace is end-group character
7061 \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).

```
7062 \openin 0 hyphen.cfg
7063 \ifeof0
7064 \else
7065   \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.

```
7066   \def\input #1 {%
7067     \let\input\a
7068     \a hyphen.cfg
7069     \let\a\undefined
7070   }
7071 \fi
7072 ⟨/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.

```
7073 ⟨bplain⟩\a plain.tex
7074 ⟨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.

```
7075 ⟨bplain⟩\def\fmtname{babel-plain}
7076 ⟨blplain⟩\def\fmtname{babel-lplain}
```

When you are using a different format, based on plain.tex you can make a copy of blplain.tex, rename it and replace plain.tex with the name of your format file.

16.2   Emulating some LaTeX features

The following code duplicates or emulates parts of LaTeX 2_ε that are needed for babel.

```
7077 ⟨⟨∗Emulate LaTeX⟩⟩ ≡
7078   % == Code for plain ==
7079 \def\@empty{}
7080 \def\loadlocalcfg#1{%
7081   \openin0#1.cfg
7082   \ifeof0
7083     \closein0
7084   \else
7085     \closein0
7086     {\immediate\write16{*********************************}%
7087     \immediate\write16{* Local config file #1.cfg used}%
7088     \immediate\write16{*}%
7089     }
7090     \input #1.cfg\relax
7091   \fi
7092   \@endofldf}
```

16.3   General tools

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

```
7093 \long\def\@firstofone#1{#1}
7094 \long\def\@firstoftwo#1#2{#1}
7095 \long\def\@secondoftwo#1#2{#2}
7096 \def\@nnil{\@nil}
7097 \def\@gobbletwo#1#2{}
7098 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
7099 \def\@star@or@long#1{%
7100   \@ifstar
```

```
7101    {\let\l@ngrel@x\relax#1}%
7102    {\let\l@ngrel@x\long#1}}
7103 \let\l@ngrel@x\relax
7104 \def\@car#1#2\@nil{#1}
7105 \def\@cdr#1#2\@nil{#2}
7106 \let\@typeset@protect\relax
7107 \let\protected@edef\edef
7108 \long\def\@gobble#1{}
7109 \edef\@backslashchar{\expandafter\@gobble\string\\}
7110 \def\strip@prefix#1>{}
7111 \def\g@addto@macro#1#2{{%
7112    \toks@\expandafter{#1#2}%
7113    \xdef#1{\the\toks@}}}
7114 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
7115 \def\@nameuse#1{\csname #1\endcsname}
7116 \def\@ifundefined#1{%
7117    \expandafter\ifx\csname#1\endcsname\relax
7118      \expandafter\@firstoftwo
7119    \else
7120      \expandafter\@secondoftwo
7121    \fi}
7122 \def\@expandtwoargs#1#2#3{%
7123    \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
7124 \def\zap@space#1 #2{%
7125    #1%
7126    \ifx#2\@empty\else\expandafter\zap@space\fi
7127    #2}
7128 \let\bbl@trace\@gobble
```

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

```
7129 \ifx\@preamblecmds\@undefined
7130    \def\@preamblecmds{}
7131 \fi
7132 \def\@onlypreamble#1{%
7133    \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
7134      \@preamblecmds\do#1}}
7135 \@onlypreamble\@onlypreamble
```

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

```
7136 \def\begindocument{%
7137    \@begindocumenthook
7138    \global\let\@begindocumenthook\@undefined
7139    \def\do##1{\global\let##1\@undefined}%
7140    \@preamblecmds
7141    \global\let\do\noexpand}
7142 \ifx\@begindocumenthook\@undefined
7143    \def\@begindocumenthook{}
7144 \fi
7145 \@onlypreamble\@begindocumenthook
7146 \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.

```
7147 \def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
7148 \@onlypreamble\AtEndOfPackage
7149 \def\@endofldf{}
7150 \@onlypreamble\@endofldf
7151 \let\bbl@afterlang\@empty
```

7152 `\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.

```
7153 \catcode`\&=\z@
7154 \ifx&if@filesw\@undefined
7155   \expandafter\let\csname if@filesw\expandafter\endcsname
7156     \csname iffalse\endcsname
7157 \fi
7158 \catcode`\&=4
```

Mimick LaTeX's commands to define control sequences.

```
7159 \def\newcommand{\@star@or@long\new@command}
7160 \def\new@command#1{%
7161   \@testopt{\@newcommand#1}0}
7162 \def\@newcommand#1[#2]{%
7163   \@ifnextchar [{\@xargdef#1[#2]}%
7164                {\@argdef#1[#2]}}
7165 \long\def\@argdef#1[#2]#3{%
7166   \@yargdef#1\@ne{#2}{#3}}
7167 \long\def\@xargdef#1[#2][#3]#4{%
7168   \expandafter\def\expandafter#1\expandafter{%
7169     \expandafter\@protected@testopt\expandafter #1%
7170     \csname\string#1\expandafter\endcsname{#3}}%
7171   \expandafter\@yargdef \csname\string#1\endcsname
7172   \tw@{#2}{#4}}
7173 \long\def\@yargdef#1#2#3{%
7174   \@tempcnta#3\relax
7175   \advance \@tempcnta \@ne
7176   \let\@hash@\relax
7177   \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
7178   \@tempcntb #2%
7179   \@whilenum\@tempcntb <\@tempcnta
7180   \do{%
7181     \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
7182     \advance\@tempcntb \@ne}%
7183   \let\@hash@##%
7184   \l@ngrel@x\expandafter\def\expandafter#1\reserved@a}
7185 \def\providecommand{\@star@or@long\provide@command}
7186 \def\provide@command#1{%
7187   \begingroup
7188     \escapechar\m@ne\xdef\@gtempa{{\string#1}}%
7189   \endgroup
7190   \expandafter\@ifundefined\@gtempa
7191     {\def\reserved@a{\new@command#1}}%
7192     {\let\reserved@a\relax
7193      \def\reserved@a{\new@command\reserved@a}}%
7194   \reserved@a}%

7195 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
7196 \def\declare@robustcommand#1{%
7197   \edef\reserved@a{\string#1}%
7198   \def\reserved@b{#1}%
7199   \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
7200   \edef#1{%
7201     \ifx\reserved@a\reserved@b
7202       \noexpand\x@protect
7203       \noexpand#1%
7204     \fi
```

```
7205        \noexpand\protect
7206        \expandafter\noexpand\csname
7207            \expandafter\@gobble\string#1 \endcsname
7208      }%
7209      \expandafter\new@command\csname
7210          \expandafter\@gobble\string#1 \endcsname
7211 }
7212 \def\x@protect#1{%
7213    \ifx\protect\@typeset@protect\else
7214        \@x@protect#1%
7215    \fi
7216 }
7217 \catcode`\&=\z@  % Trick to hide conditionals
7218    \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`.

```
7219    \def\bbl@tempa{\csname newif\endcsname&ifin@}
7220 \catcode`\&=4
7221 \ifx\in@\@undefined
7222    \def\in@#1#2{%
7223      \def\in@@##1#1##2##3\in@@{%
7224        \ifx\in@##2\in@false\else\in@true\fi}%
7225      \in@@#2#1\in@\in@@}
7226 \else
7227    \let\bbl@tempa\@empty
7228 \fi
7229 \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).

```
7230 \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.

```
7231 \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.

```
7232 \ifx\@tempcnta\@undefined
7233    \csname newcount\endcsname\@tempcnta\relax
7234 \fi
7235 \ifx\@tempcntb\@undefined
7236    \csname newcount\endcsname\@tempcntb\relax
7237 \fi
```

To prevent wasting two counters in LaTeX 2.09 (because counters with the same name are allocated later by it) we reset the counter that holds the next free counter (`\count10`).

```
7238 \ifx\bye\@undefined
7239    \advance\count10 by -2\relax
7240 \fi
7241 \ifx\@ifnextchar\@undefined
7242    \def\@ifnextchar#1#2#3{%
7243      \let\reserved@d=#1%
7244      \def\reserved@a{#2}\def\reserved@b{#3}%
```

215

```
7245        \futurelet\@let@token\@ifnch}
7246   \def\@ifnch{%
7247        \ifx\@let@token\@sptoken
7248          \let\reserved@c\@xifnch
7249        \else
7250          \ifx\@let@token\reserved@d
7251            \let\reserved@c\reserved@a
7252          \else
7253            \let\reserved@c\reserved@b
7254          \fi
7255        \fi
7256        \reserved@c}
7257   \def\:{\let\@sptoken= } \:  % this makes \@sptoken a space token
7258   \def\:{\@xifnch} \expandafter\def\: {\futurelet\@let@token\@ifnch}
7259 \fi
7260 \def\@testopt#1#2{%
7261   \@ifnextchar[{#1}{#1[{#2}]}}
7262 \def\@protected@testopt#1{%
7263   \ifx\protect\@typeset@protect
7264     \expandafter\@testopt
7265   \else
7266     \@x@protect#1%
7267   \fi}
7268 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
7269       #2\relax}\fi}
7270 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
7271         \else\expandafter\@gobble\fi{#1}}
```

16.4   Encoding related macros

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

```
7272 \def\DeclareTextCommand{%
7273   \@dec@text@cmd\providecommand
7274 }
7275 \def\ProvideTextCommand{%
7276   \@dec@text@cmd\providecommand
7277 }
7278 \def\DeclareTextSymbol#1#2#3{%
7279   \@dec@text@cmd\chardef#1{#2}#3\relax
7280 }
7281 \def\@dec@text@cmd#1#2#3{%
7282   \expandafter\def\expandafter#2%
7283       \expandafter{%
7284         \csname#3-cmd\expandafter\endcsname
7285         \expandafter#2%
7286         \csname#3\string#2\endcsname
7287       }%
7288 %   \let\@ifdefinable\@rc@ifdefinable
7289   \expandafter#1\csname#3\string#2\endcsname
7290 }
7291 \def\@current@cmd#1{%
7292   \ifx\protect\@typeset@protect\else
7293       \noexpand#1\expandafter\@gobble
7294   \fi
7295 }
7296 \def\@changed@cmd#1#2{%
7297   \ifx\protect\@typeset@protect
7298       \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
```

```
7299        \expandafter\ifx\csname ?\string#1\endcsname\relax
7300          \expandafter\def\csname ?\string#1\endcsname{%
7301            \@changed@x@err{#1}%
7302          }%
7303        \fi
7304        \global\expandafter\let
7305          \csname\cf@encoding \string#1\expandafter\endcsname
7306          \csname ?\string#1\endcsname
7307      \fi
7308      \csname\cf@encoding\string#1%
7309        \expandafter\endcsname
7310    \else
7311      \noexpand#1%
7312    \fi
7313 }
7314 \def\@changed@x@err#1{%
7315    \errhelp{Your command will be ignored, type <return> to proceed}%
7316    \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
7317 \def\DeclareTextCommandDefault#1{%
7318    \DeclareTextCommand#1?%
7319 }
7320 \def\ProvideTextCommandDefault#1{%
7321    \ProvideTextCommand#1?%
7322 }
7323 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
7324 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
7325 \def\DeclareTextAccent#1#2#3{%
7326   \DeclareTextCommand#1{#2}[1]{\accent#3 ##1}
7327 }
7328 \def\DeclareTextCompositeCommand#1#2#3#4{%
7329    \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
7330    \edef\reserved@b{\string##1}%
7331    \edef\reserved@c{%
7332      \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
7333    \ifx\reserved@b\reserved@c
7334      \expandafter\expandafter\expandafter\ifx
7335        \expandafter\@car\reserved@a\relax\relax\@nil
7336        \@text@composite
7337      \else
7338        \edef\reserved@b##1{%
7339          \def\expandafter\noexpand
7340            \csname#2\string#1\endcsname####1{%
7341            \noexpand\@text@composite
7342              \expandafter\noexpand\csname#2\string#1\endcsname
7343              ####1\noexpand\@empty\noexpand\@text@composite
7344              {##1}%
7345          }%
7346        }%
7347        \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
7348      \fi
7349      \expandafter\def\csname\expandafter\string\csname
7350        #2\endcsname\string#1-\string#3\endcsname{#4}
7351    \else
7352      \errhelp{Your command will be ignored, type <return> to proceed}%
7353      \errmessage{\string\DeclareTextCompositeCommand\space used on
7354          inappropriate command \protect#1}
7355    \fi
7356 }
7357 \def\@text@composite#1#2#3\@text@composite{%
```

```
7358    \expandafter\@text@composite@x
7359        \csname\string#1-\string#2\endcsname
7360 }
7361 \def\@text@composite@x#1#2{%
7362    \ifx#1\relax
7363        #2%
7364    \else
7365        #1%
7366    \fi
7367 }
7368 %
7369 \def\@strip@args#1:#2-#3\@strip@args{#2}
7370 \def\DeclareTextComposite#1#2#3#4{%
7371    \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
7372    \bgroup
7373        \lccode`\@=#4%
7374        \lowercase{%
7375    \egroup
7376        \reserved@a @%
7377    }%
7378 }
7379 %
7380 \def\UseTextSymbol#1#2{#2}
7381 \def\UseTextAccent#1#2#3{}
7382 \def\@use@text@encoding#1{}
7383 \def\DeclareTextSymbolDefault#1#2{%
7384    \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
7385 }
7386 \def\DeclareTextAccentDefault#1#2{%
7387    \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
7388 }
7389 \def\cf@encoding{OT1}
```

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

```
7390 \DeclareTextAccent{\"}{OT1}{127}
7391 \DeclareTextAccent{\'}{OT1}{19}
7392 \DeclareTextAccent{\^}{OT1}{94}
7393 \DeclareTextAccent{\`}{OT1}{18}
7394 \DeclareTextAccent{\~}{OT1}{126}
```

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

```
7395 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
7396 \DeclareTextSymbol{\textquotedblright}{OT1}{`\"}
7397 \DeclareTextSymbol{\textquoteleft}{OT1}{`\`}
7398 \DeclareTextSymbol{\textquoteright}{OT1}{`\'}
7399 \DeclareTextSymbol{\i}{OT1}{16}
7400 \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.

```
7401 \ifx\scriptsize\@undefined
7402    \let\scriptsize\sevenrm
7403 \fi
7404    % End of code for plain
7405 ⟨⟨/Emulate LaTeX⟩⟩
```

A proxy file:

```
7406 ⟨*plain⟩
7407 \input babel.def
7408 ⟨/plain⟩
```

218

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).