Babel

Johannes L. Braams
Original author

Javier Bezos
Current maintainer

Localization and internationalization

Unicode
T_EX
pdfT_EX
LuaT_EX
XeT_EX

Contents

1

Troubleshoooting

Part I

User guide

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

What if I'm interested only in the latest changes? Changes and new features with relation to version 3.8 are highlighted with ~~New X.XX~~, and there are some notes for the latest versions in the babel 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.

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".

```
{\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

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

14

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.

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}
```

18

```
\begin{document}

\tableofcontents

\chapter{სამზარეულო და სუფრის ტრადიციები}

ქართული ტრადიციული სამზარეულო ერთ-ერთი უმდიდრესია მთელ მსოფლიოში.

\end{document}
```

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

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

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

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

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

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

Or also:

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

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

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

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

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

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

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

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

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

```
\babelprovide[import, hyphenrules=+]{lao}
\babelpatterns[lao]{1ກ 1�ມ 1ສ 1ງ 1ກ 1ຈ} % Random
```

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

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

Latin, Greek, Cyrillic Combining chars with the default luatex font renderer might be wrong; on then other hand, with the Harfbuzz renderer diacritics are stacked correctly, but many hyphenations points are discarded (this bug 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

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

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

aghem
akan
albanian
american
amharic
ancientgreek
arabic
arabic-algeria
arabic-DZ
arabic-morocco
arabic-MA
arabic-syria
arabic-SY
armenian
assamese
asturian
asu
australian
austrian
azerbaijani-cyrillic
azerbaijani-cyrl
azerbaijani-latin
azerbaijani-latn
azerbaijani
bafia
bambara
basaa
basque
belarusian
bemba
bena
bengali
bodo
bosnian-cyrillic
bosnian-cyrl
bosnian-latin
bosnian-latn
bosnian
brazilian
breton
british
bulgarian
burmese
canadian

cantonese
catalan
centralatlastamazight
centralkurdish
chechen
cherokee
chiga
chinese-hans-hk
chinese-hans-mo
chinese-hans-sg
chinese-hans
chinese-hant-hk
chinese-hant-mo
chinese-hant
chinese-simplified-hongkongsarchina
chinese-simplified-macausarchina
chinese-simplified-singapore
chinese-simplified
chinese-traditional-hongkongsarchina
chinese-traditional-macausarchina
chinese-traditional
chinese
churchslavic
churchslavic-cyrs
churchslavic-oldcyrillic[12]
churchsslavic-glag
churchsslavic-glagolitic
colognian
cornish
croatian
czech
danish
duala
dutch
dzongkha
embu
english-au
english-australia
english-ca
english-canada
english-gb
english-newzealand
english-nz
english-unitedkingdom

[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.

```
\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.

captions=   *⟨language-tag⟩*

Loads only the strings. For example:

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

hyphenrules=   *⟨language-list⟩*

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

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

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

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

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

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

EXAMPLE Let's assume your document is mainly in Polytonic Greek, but with some sections in Italian. Then, the first attempt should be:

```
\usepackage[italian, greek.polutonic]{babel}
```

But if, say, accents in Greek are not shown correctly, you can try:

```
\usepackage[italian]{babel}
\babelprovide[import, main]{polytonicgreek}
```

Remerber there is an alternative syntax for the latter:

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

script=   *⟨script-name⟩*

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

<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).

justification= kashida | elongated | unhyphenated

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

linebreaking= New 3.59 Just a synonymous for justification.

mapfont= direction

Assigns the font for the writing direction of this language (only with bidi=basic). Whenever possible, instead of this option use onchar, based on the script, which usually

</dl>

makes more sense. More precisely, what `mapfont=direction` means is, 'when a character has the same direction as the script for the "provided" language, then change its font to that set for this language'. There are 3 directions, following the bidi Unicode algorithm, namely, Arabic-like, Hebrew-like and left to right. So, there should be at most 3 directives of this kind.

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

1.17 Digits and counters

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

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

Languages providing native digits in all or some variants are:

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

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

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

New 4.41   Many 'ini' locale files has been extended with information about non-positional numerical systems, based on those predefined in CSS. They only work with xetex and luatex and are fully expendable (even inside an unprotected `\edef`). Currently, they are limited to numbers below 10000.
There are several ways to use them (for the availabe styles in each language, see the list below):

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

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

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

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

The styles are:

Ancient Greek lower.ancient, upper.ancient
Amharic afar, agaw, ari, blin, dizi, gedeo, gumuz, hadiyya, harari, kaffa, kebena, kembata, konso, kunama, meen, oromo, saho, sidama, silti, tigre, wolaita, yemsa
Arabic abjad, maghrebi.abjad
Belarusan, Bulgarian, Macedonian, Serbian lower, upper
Bengali alphabetic
Coptic epact, lower.letters
Hebrew letters (neither geresh nor gershayim yet)
Hindi alphabetic
Armenian lower.letter, upper.letter
Japanese hiragana, hiragana.iroha, katakana, katakana.iroha, circled.katakana, informal, formal, cjk-earthly-branch, cjk-heavenly-stem, fullwidth.lower.alpha, fullwidth.upper.alpha
Georgian letters
Greek lower.modern, upper.modern, lower.ancient, upper.ancient (all with keraia)
Khmer consonant
Korean consonant, syllabe, hanja.informal, hanja.formal, hangul.formal, cjk-earthly-branch, cjk-heavenly-stem, fullwidth.lower.alpha, fullwidth.upper.alpha
Marathi alphabetic
Persian abjad, alphabetic
Russian lower, lower.full, upper, upper.full
Syriac letters
Tamil ancient
Thai alphabetic
Ukrainian lower, lower.full, upper, upper.full
Chinese cjk-earthly-branch, cjk-heavenly-stem, fullwidth.lower.alpha, fullwidth.upper.alpha

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

1.18   Dates

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

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

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

1.19 Accessing language info

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

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

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

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

`\localeinfo` `{⟨field⟩}`

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

`name.english` as provided by the Unicode CLDR.
`tag.ini` is the tag of the `ini` file (the way this file is identified in its name).
`tag.bcp47` is the full BCP 47 tag (see the warning below).
`language.tag.bcp47` is the BCP 47 language tag.
`tag.opentype` is the tag used by OpenType (usually, but not always, the same as BCP 47).
`script.name` , as provided by the Unicode CLDR.
`script.tag.bcp47` is the BCP 47 tag of the script used by this locale.
`script.tag.opentype` is the tag used by OpenType (usually, but not always, the same as BCP 47).

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

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

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

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

the macro `\hechap` will contain the string פרק.
If the key does not exist, the macro is set to `\relax` and an error is raised. New 3.47 With the starred version no error is raised, so that you can take your own actions with undefined properties.
Babel remembers which `ini` files have been loaded. There is a loop named `\LocaleForEach` to traverse the list, where #1 is the name of the current item, so that `\LocaleForEach{\message{ **#1** }}` just shows the loaded `ini`'s.

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

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

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

1.20 Hyphenation and line breaking

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

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

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

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

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

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

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

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

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

glue >0 pt (at the beginning of a word, provided it is not immediately preceded by, say, a parenthesis).

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

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

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

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

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

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

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

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

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

New 3.9m   *In luatex only,*[14] adds or replaces patterns for the languages given or, without the optional argument, for *all* languages. If a pattern for a certain combination already exists, it gets replaced by the new one.
It can be used only in the preamble, and patterns are added when the language is first selected, thus taking into account changes of \lccodes's done in \extras⟨*lang*⟩ as well as the language-specific encoding (not set in the preamble by default). Multiple \babelpatterns's are allowed.
Listed patterns are saved expanded and therefore it relies on the LICR. Of course, it also works without the LICR if the input and the font encodings are the same, like in Unicode based engines.
New 3.31   (Only luatex.) With \babelprovide and imported CJK languages, a simple generic line breaking algorithm (push-out-first) is applied, based on a selection of the Unicode rules (New 3.32   it is disabled in verbatim mode, or more precisely when the

---

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

hyphenrules are set to nohyphenation). It can be activated alternatively by setting explicitly the `intraspace`.

New 3.27 Interword spacing for Thai, Lao and Khemer is activated automatically if a language with one of those scripts are loaded with `\babelprovide`. See the sample on the babel repository. With both Unicode engines, spacing is based on the "current" em unit (the size of the previous char in luatex, and the font size set by the last `\selectfont` in xetex).

1.21 Transforms

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

It currently embraces `\babelprehyphenation` and `\babelposthyphenation`.

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

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

Here are the transforms currently predefined. (More to follow in future releases.)

Arabic	`transliteration.dad`	Applies the transliteration system devised by Yannis Haralambous for dad (simple and TeX-friendly). Not yet complete, but sufficient for most texts.
Croatian	`digraphs.ligatures`	Ligatures *DŽ, Dž, dž, LJ, Lj, lj, NJ, Nj, nj*. It assumes they exist. This is not the recommended way to make these transformations (the best way is with OTF features), but it can get you out of a hurry.
Czech, Polish, Portuguese, Slovak, Spanish	`hyphen.repeat`	Explicit hyphens behave like `\babelhyphen{repeat}`.
Czech, Polish, Slovak	`oneletter.nobreak`	Converts a space after a non-syllabic preposition or conjunction into a non-breaking space.
Greek	`diaeresis.hyphen`	Removes the diaeresis above iota and upsilon if hyphenated just before. It works with the three variants.
Hindi, Sanskrit	`transliteration.hk`	The Harvard-Kyoto system to romanize Devanagari.
Hindi, Sanskrit	`punctuation.space`	Inserts a space before the following four characters: *!?:;*.
Hungarian	`digraphs.hyphen`	Hyphenates the long digraphs *ccs, ddz, ggy, lly, nny, ssz, tty* and *zzs* as *cs-cs, dz-dz*, etc.

---

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

Arabic, Persian	`kashida.plain`	Experimental. A very simple and basic transform for 'plain' Arabic fonts, which attempts to distribute the tatwil as evenly as possible (starting at the end of the line). See the news for version 3.59.
Serbian	`transliteration.gajica`	(Note `serbian` with `ini` files refers to the Cyrillic script, which is here the target.) The standard system devised by Ljudevit Gaj.

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

New 3.37-3.39 *With luatex* it is 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. Only a few rules are currently provided (see below), but they can be defined as shown in the following example, where {1} is the first captured char (between ( ) in the pattern):

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

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

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

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

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

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

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

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

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

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

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

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

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

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

1.22   Selection based on BCP 47 tags

New 3.43   The recommended way to select languages is that described at the beginning of this document. However, BCP 47 tags are becoming customary, particularly in documents (or parts of documents) generated by external sources, and therefore babel will provide a set of tools to select the locales in different situations, adapted to the particular needs of each case. Currently, babel provides autoloading of locales as described in this section. In these contexts autoloading is particularly important because we may not know on beforehand which languages will be requested.
It must be activated explicitly, because it is primarily meant for special tasks. Mapping from BCP 47 codes to locale names are not hardcoded in babel. Instead the data is taken from the ini files, which means currently about 250 tags are already recognized. Babel performs a simple lookup in the following way: `fr-Latn-FR` → `fr-Latn` → `fr-FR` → `fr`. Languages with the same resolved name are considered the same. Case is normalized before, so that `fr-latn-fr` → `fr-Latn-FR`. If a tag and a name overlap, the tag takes precedence.
Here is a minimal example:

```
\documentclass{article}

\usepackage[danish]{babel}

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

\begin{document}

Chapter in Danish: \chaptername.

\selectlanguage{de-AT}

\localedate{2020}{1}{30}

\end{document}
```

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

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

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

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

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

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

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

1.23   Selecting scripts

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

`\ensureascii`   {⟨*text*⟩}

New 3.9i   This macro makes sure ⟨*text*⟩ is typeset with a LICR-savvy encoding in the ASCII range. It is used to redefine `\TeX` and `\LaTeX` so that they are correctly typeset even with LGR or X2 (the complete list is stored in `\BabelNonASCII`, which by default is LGR, X2, OT2, OT3, OT6, LHE, LWN, LMA, LMC, LMS, LMU, but you can modify it). So, in some sense it fixes the bug described in the previous paragraph.
If non-ASCII encodings are not loaded (or no encoding at all), it is no-op (also `\TeX` and `\LaTeX` are not redefined); otherwise, `\ensureascii` switches to the encoding at the beginning of the document if ASCII-savvy, or else the last ASCII-savvy encoding loaded. For example, if you load LY1,LGR, then it is set to LY1, but if you load LY1,T2A it is set to T2A. The symbol encodings TS1, T3, and TS3 are not taken into account, since they are not used

---

[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.

for "ordinary" text (they are stored in \BabelNonText, used in some special cases when no Latin encoding is explicitly set).

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

1.24 Selecting directions

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

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

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

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

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

There are some package options controlling bidi writing.

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

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

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

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

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

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

```
\documentclass{article}

\usepackage[bidi=basic]{babel}
```

```
\babelprovide[import, main]{arabic}

\babelfont{rm}{FreeSerif}

\begin{document}

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

\end{document}
```

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

```
\documentclass{book}

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

\babelprovide[onchar=ids fonts]{arabic}

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

\begin{document}

Most Arabic speakers consider the two varieties to be two registers
of one language, although the two registers can be referred to in
Arabic as فصحى العصر \textit{fuṣḥā l-ʻaṣr} (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]

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*⟩}

---

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

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:

45

```
    \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

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.
`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*⟩}

---

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

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

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

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

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

1.29   Tweaking some features

\babeladjust   {⟨*key-value-list*⟩}

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

1.30   Tips, workarounds, known issues and notes

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

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

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

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

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

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

- For the hyphenation to work correctly, lccodes cannot change, because T_EX 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 T_EX, 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.

- 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 T_EX 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 L^AT_EX 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.

---

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

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

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

1.32 Tentative and experimental code

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

Options for locales loaded on the fly

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

Labels

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

2   Loading languages with `language.dat`

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

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

2.1   Format

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

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

---

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

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

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

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

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

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

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

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

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

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

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

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

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

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

- The language definition files must define five macros, used to activate and deactivate the language-specific definitions. These macros are \⟨*lang*⟩hyphenmins, \captions⟨*lang*⟩, \date⟨*lang*⟩, \extras⟨*lang*⟩ and \noextras⟨*lang*⟩(the last two may be left empty); where ⟨*lang*⟩ is either the name of the language definition file or the name of the LaTeX option that is to be used. These macros and their functions are

---

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

discussed below. You must define all or none for a language (or a dialect); defining, say, \date⟨*lang*⟩ but not \captions⟨*lang*⟩ does not raise an error but can lead to unexpected results.

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

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

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

Some recommendations:

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

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

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

- Do not switch scripts. If you want to make sure a set of glyphs is used, switch either the font encoding (low-level) or the language (high-level, which in turn may switch the font encoding). Usage of things like \latintext is deprecated.[26]

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

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

3.1 Guidelines for contributed languages

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

---

[26]But not removed, for backward compatibility.

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

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

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

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

The following page provides a starting point for `ldf` files: `http://www.texnia.com/incubator.html`. See also `https://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,
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.

---

[27]This mechanism was introduced by Bernd Raichle.

3.8 Encoding-dependent strings

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

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

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

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

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

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

A "selector" is a name to be used as value in package option `strings`, optionally followed by extra info about the encodings to be used. The name `unicode` must be used for xetex and luatex (the key `strings` has also other two special values: `generic` and `encoded`).

If a string is set several times (because several blocks are read), the first one takes precedence (ie, it works much like `\providecommand`).

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

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

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

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

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

---

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

```
\SetString{\chaptername}{utf8-string}

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

\EndBabelCommands
```

A real example is:

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

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

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

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

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

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

\EndBabelCommands
```

When used in ldf files, previous values of \⟨*category*⟩⟨*language*⟩ are overridden, which means the old way to define strings still works and used by default (to be precise, is first set 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

59

maintainers of the current languages to decide if using it is appropriate.[29]

\EndBabelCommands   Marks the end of the series of blocks.

\AfterBabelCommands   {⟨*code*⟩}

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

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

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

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

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

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

#1 is replaced by the roman numeral.

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

Sets globally code to be executed at \MakeUppercase and \MakeLowercase. The code would typically be things like \let\BB\bb and \uccode or \lccode (although for the reasons explained above, changes in lc/uc codes may not work). A ⟨*map-list*⟩ is a series of macros using the internal format of \@uclclist (eg, \bb\BB\cc\CC). The mandatory 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}
```

---

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

```
\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 encondings.
[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=3.59.2379⟩⟩
2 ⟨⟨date=2021/05/21⟩⟩

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}
```

---

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

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
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 LaTeX forgets 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
482      \fi}
483 \fi
```

`\ref`
`\pageref` The same holds for the macro `\ref` that references a label and `\pageref` to reference a page. We 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@}}}}}%
```

\markboth The definition of \markboth is equivalent to that of \markright, except that we need two token \@mkboth registers. The documentclasses report and book define and set the headings for the page. While doing so they also store a copy of \markboth in \@mkboth. Therefore we need to check whether \@mkboth has already been set. If so we neeed to do that again with the new definition of \markboth. (As of Oct 2019, LaTeX stores the definition in an intermediate macro, so it's not necessary anymore, but it's preserved for older versions.)

```
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`

`\ifthenelse` Sometimes a document writer wants to create a special effect depending on the page a certain fragment of text appears on. This can be achieved by the following piece of code:

```
\ifthenelse{\isodd{\pageref{some:label}}}
           {code for odd pages}
           {code for even pages}
```

In order for this to work the argument of `\isodd` needs to be fully expandable. With the above redefinition of `\pageref` it is not in the case of this example. To overcome that, we add some code to the definition of `\ifthenelse` to make things work.

We want to revert the definition of `\pageref` and `\ref` to their original definition for the first argument of `\ifthenelse`, so we first need to store their current meanings.

Then we can set the `\@safe@actives` switch and call the original `\ifthenelse`. In order to be able to use shorthands in the second and third arguments of `\ifthenelse` the resetting of the switch *and* the definition of `\pageref` happens inside those arguments.

```
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.

```
643 \bbl@trace{Encoding and fonts}
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#1ENC.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
```

79

```
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
```

81

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
```

84

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

94

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}%
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   \else
1520     % other = select patterns
1521     \bbl@patterns{#1}%
1522   \fi
1523   % hyphenation - mins
1524   \babel@savevariable\lefthyphenmin
1525   \babel@savevariable\righthyphenmin
1526   \expandafter\ifx\csname #1hyphenmins\endcsname\relax
1527     \set@hyphenmins\tw@\thr@@\relax
1528   \else
1529     \expandafter\expandafter\expandafter\set@hyphenmins
1530       \csname #1hyphenmins\endcsname\relax
1531   \fi}
```

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

```
1532 \long\def\otherlanguage#1{%
1533   \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\thr@@\fi
1534   \csname selectlanguage \endcsname{#1}%
1535   \ignorespaces}
```

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

```
1536 \long\def\endotherlanguage{%
1537   \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`.

```
1538 \expandafter\def\csname otherlanguage*\endcsname{%
1539   \@ifnextchar[\bbl@otherlanguage@s{\bbl@otherlanguage@s[]}}
1540 \def\bbl@otherlanguage@s[#1]#2{%
1541   \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
1542   \def\bbl@select@opts{#1}%
1543   \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".

```
1544 \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

$\verb|\extras|\langle lang\rangle$ command doesn't make any $\verb|\global|$ changes. The coding is very similar to part of $\verb|\selectlanguage|$.

$\verb|\bbl@beforeforeign|$ is a trick to fix a bug in bidi texts. $\verb|\foreignlanguage|$ is supposed to be a 'text' command, and therefore it must emit a $\verb|\leavevmode|$, 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) $\verb|\foreignlanguage*|$ is a temporary, experimental macro for a few lines with a different script direction, while preserving the paragraph format (thank the braces around $\verb|\par|$, things like $\verb|\hangindent|$ 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 $\verb|\BabelText|$ 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 $\verb|\foreignlanguage|$ enters into hmode with the surrounding lang, and with $\verb|\foreignlanguage*|$ with the new lang.

```
1545 \providecommand\bbl@beforeforeign{}
1546 \edef\foreignlanguage{%
1547   \noexpand\protect
1548   \expandafter\noexpand\csname foreignlanguage \endcsname}
1549 \expandafter\def\csname foreignlanguage \endcsname{%
1550   \@ifstar\bbl@foreign@s\bbl@foreign@x}
1551 \providecommand\bbl@foreign@x[3][]{%
1552   \begingroup
1553     \def\bbl@select@opts{#1}%
1554     \let\BabelText\@firstofone
1555     \bbl@beforeforeign
1556     \foreign@language{#2}%
1557     \bbl@usehooks{foreign}{}%
1558     \BabelText{#3}% Now in horizontal mode!
1559   \endgroup}
1560 \def\bbl@foreign@s#1#2{% TODO - \shapemode, \@setpar, ?\@@par
1561   \begingroup
1562     {\par}%
1563     \let\bbl@select@opts\@empty
1564     \let\BabelText\@firstofone
1565     \foreign@language{#1}%
1566     \bbl@usehooks{foreign*}{}%
1567     \bbl@dirparastext
1568     \BabelText{#2}% Still in vertical mode!
1569     {\par}%
1570   \endgroup}
```

$\verb|\foreign@language|$ This macro does the work for $\verb|\foreignlanguage|$ 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.

```
1571 \def\foreign@language#1{%
1572   % set name
1573   \edef\languagename{#1}%
1574   \ifbbl@usedategroup
1575     \bbl@add\bbl@select@opts{,date,}%
1576     \bbl@usedategroupfalse
1577   \fi
1578   \bbl@fixname\languagename
1579   % TODO. name@map here?
1580   \bbl@provide@locale
1581   \bbl@iflanguage\languagename{%
1582     \expandafter\ifx\csname date\languagename\endcsname\relax
```

```
1583        \bbl@warning    % TODO - why a warning, not an error?
1584          {Unknown language `#1'. Either you have\\%
1585           misspelled its name, it has not been installed,\\%
1586           or you requested it in a previous run. Fix its name,\\%
1587           install it or just rerun the file, respectively. In\\%
1588           some cases, you may need to remove the aux file.\\%
1589           I'll proceed, but expect wrong results.\\%
1590           Reported}%
1591      \fi
1592      % set type
1593      \let\bbl@select@type\@ne
1594      \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.

```
1595 \let\bbl@hyphlist\@empty
1596 \let\bbl@hyphenation@\relax
1597 \let\bbl@pttnlist\@empty
1598 \let\bbl@patterns@\relax
1599 \let\bbl@hymapsel=\@cclv
1600 \def\bbl@patterns#1{%
1601   \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
1602       \csname l@#1\endcsname
1603       \edef\bbl@tempa{#1}%
1604     \else
1605       \csname l@#1:\f@encoding\endcsname
1606       \edef\bbl@tempa{#1:\f@encoding}%
1607     \fi
1608   \@expandtwoargs\bbl@usehooks{patterns}{{#1}{\bbl@tempa}}%
1609   % > luatex
1610   \@ifundefined{bbl@hyphenation@}{}{% Can be \relax!
1611     \begingroup
1612       \bbl@xin@{,\number\language,}{,\bbl@hyphlist}%
1613       \ifin@\else
1614         \@expandtwoargs\bbl@usehooks{hyphenation}{{#1}{\bbl@tempa}}%
1615         \hyphenation{%
1616           \bbl@hyphenation@
1617           \@ifundefined{bbl@hyphenation@#1}%
1618             \@empty
1619             {\space\csname bbl@hyphenation@#1\endcsname}}%
1620         \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
1621       \fi
1622     \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*.

```
1623 \def\hyphenrules#1{%
1624   \edef\bbl@tempf{#1}%
1625   \bbl@fixname\bbl@tempf
1626   \bbl@iflanguage\bbl@tempf{%
```

```
1627        \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
1628        \ifx\languageshorthands\@undefined\else
1629          \languageshorthands{none}%
1630        \fi
1631        \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax
1632          \set@hyphenmins\tw@\thr@@\relax
1633        \else
1634          \expandafter\expandafter\expandafter\set@hyphenmins
1635          \csname\bbl@tempf hyphenmins\endcsname\relax
1636        \fi}}
1637 \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.

```
1638 \def\providehyphenmins#1#2{%
1639    \expandafter\ifx\csname #1hyphenmins\endcsname\relax
1640      \@namedef{#1hyphenmins}{#2}%
1641    \fi}
```

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

```
1642 \def\set@hyphenmins#1#2{%
1643    \lefthyphenmin#1\relax
1644    \righthyphenmin#2\relax}
```

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

```
1645 \ifx\ProvidesFile\@undefined
1646    \def\ProvidesLanguage#1[#2 #3 #4]{%
1647      \wlog{Language: #1 #4 #3 <#2>}%
1648      }
1649 \else
1650    \def\ProvidesLanguage#1{%
1651      \begingroup
1652        \catcode`\ 10 %
1653        \@makeother\/%
1654        \@ifnextchar[%
1655          {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}}
1656    \def\@provideslanguage#1[#2]{%
1657      \wlog{Language: #1 #2}%
1658      \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
1659      \endgroup}
1660 \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.

```
1661 \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.

```
1662 \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':

```
1663 \providecommand\setlocale{%
1664    \bbl@error
```

```
1665        {Not yet available}%
1666        {Find an armchair, sit down and wait}}
1667 \let\uselocale\setlocale
1668 \let\locale\setlocale
1669 \let\selectlocale\setlocale
1670 \let\localename\setlocale
1671 \let\textlocale\setlocale
1672 \let\textlanguage\setlocale
1673 \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 LaTeX 2_ε, so we can safely use its error
handling interface. Otherwise we'll have to 'keep it simple'.
Infos are not written to the console, but on the other hand many people think warnings are errors, so
a further message type is defined: an important info which is sent to the console.

```
1674 \edef\bbl@nulllanguage{\string\language=0}
1675 \ifx\PackageError\@undefined  % TODO. Move to Plain
1676   \def\bbl@error#1#2{%
1677     \begingroup
1678       \newlinechar=`\^^J
1679       \def\\{^^J(babel) }%
1680       \errhelp{#2}\errmessage{\\#1}%
1681     \endgroup}
1682   \def\bbl@warning#1{%
1683     \begingroup
1684       \newlinechar=`\^^J
1685       \def\\{^^J(babel) }%
1686       \message{\\#1}%
1687     \endgroup}
1688   \let\bbl@infowarn\bbl@warning
1689   \def\bbl@info#1{%
1690     \begingroup
1691       \newlinechar=`\^^J
1692       \def\\{^^J}%
1693       \wlog{#1}%
1694     \endgroup}
1695 \fi
1696 \def\bbl@nocaption{\protect\bbl@nocaption@i}
1697 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
1698   \global\@namedef{#2}{\textbf{?#1?}}%
1699   \@nameuse{#2}%
1700   \edef\bbl@tempa{#1}%
1701   \bbl@sreplace\bbl@tempa{name}{}%
1702   \bbl@warning{% TODO.
1703     \@backslashchar#1 not set for '\languagename'. Please,\\%
1704     define it after the language has been loaded\\%
1705     (typically in the preamble) with:\\%
1706     \string\setlocalecaption{\languagename}{\bbl@tempa}{..}\\%
1707     Reported}}
1708 \def\bbl@tentative{\protect\bbl@tentative@i}
1709 \def\bbl@tentative@i#1{%
```

```
1710    \bbl@warning{%
1711      Some functions for '#1' are tentative.\\%
1712      They might not work as expected and their behavior\\%
1713      could change in the future.\\%
1714      Reported}}
1715 \def\@nolanerr#1{%
1716    \bbl@error
1717      {You haven't defined the language #1\space yet.\\%
1718       Perhaps you misspelled it or your installation\\%
1719       is not complete}%
1720      {Your command will be ignored, type <return> to proceed}}
1721 \def\@nopatterns#1{%
1722    \bbl@warning
1723      {No hyphenation patterns were preloaded for\\%
1724       the language `#1' into the format.\\%
1725       Please, configure your TeX system to add them and\\%
1726       rebuild the format. Now I will use the patterns\\%
1727       preloaded for \bbl@nulllanguage\space instead}}
1728 \let\bbl@usehooks\@gobbletwo
1729 \ifx\bbl@onlyswitch\@empty\endinput\fi
1730   % Here ended switch.def

 Here ended switch.def.

1731 \ifx\directlua\@undefined\else
1732   \ifx\bbl@luapatterns\@undefined
1733     \input luababel.def
1734   \fi
1735 \fi
1736 ⟨⟨Basic macros⟩⟩
1737 \bbl@trace{Compatibility with language.def}
1738 \ifx\bbl@languages\@undefined
1739   \ifx\directlua\@undefined
1740     \openin1 = language.def % TODO. Remove hardcoded number
1741     \ifeof1
1742       \closein1
1743       \message{I couldn't find the file language.def}
1744     \else
1745       \closein1
1746       \begingroup
1747         \def\addlanguage#1#2#3#4#5{%
1748           \expandafter\ifx\csname lang@#1\endcsname\relax\else
1749             \global\expandafter\let\csname l@#1\expandafter\endcsname
1750               \csname lang@#1\endcsname
1751           \fi}%
1752         \def\uselanguage#1{}%
1753         \input language.def
1754       \endgroup
1755     \fi
1756   \fi
1757   \chardef\l@english\z@
1758 \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.

```
1759 \def\addto#1#2{%
1760   \ifx#1\@undefined
1761     \def#1{#2}%
```

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

```
1770  \def\bbl@withactive#1#2{%
1771    \begingroup
1772      \lccode`~=`#2\relax
1773      \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`.

```
1774  \def\bbl@redefine#1{%
1775    \edef\bbl@tempa{\bbl@stripslash#1}%
1776    \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1777    \expandafter\def\csname\bbl@tempa\endcsname}
1778  \@onlypreamble\bbl@redefine
```

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

```
1779  \def\bbl@redefine@long#1{%
1780    \edef\bbl@tempa{\bbl@stripslash#1}%
1781    \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1782    \expandafter\long\expandafter\def\csname\bbl@tempa\endcsname}
1783  \@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␣`.

```
1784  \def\bbl@redefinerobust#1{%
1785    \edef\bbl@tempa{\bbl@stripslash#1}%
1786    \bbl@ifunset{\bbl@tempa\space}%
1787      {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
1788       \bbl@exp{\def\\#1{\\\protect\<\bbl@tempa\space>}}}%
1789      {\bbl@exp{\let\<org@\bbl@tempa>\<\bbl@tempa\space>}}%
1790      \@namedef{\bbl@tempa\space}}
1791  \@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.

```
1792  \bbl@trace{Hooks}
1793  \newcommand\AddBabelHook[3][]{%
1794    \bbl@ifunset{bbl@hk@#2}{\EnableBabelHook{#2}}{}%
1795    \def\bbl@tempa##1,#3=##2,##3\@empty{\def\bbl@tempb{##2}}%
1796    \expandafter\bbl@tempa\bbl@evargs,#3=,\@empty
```

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

```
1812 \def\bbl@evargs{,% <- don't delete this comma
1813    everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
1814    adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
1815    beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
1816    hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%
1817    beforestart=0,languagename=2}
```

<code>\babelensure</code> 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.

```
1818 \bbl@trace{Defining babelensure}
1819 \newcommand\babelensure[2][]{%  TODO - revise test files
1820    \AddBabelHook{babel-ensure}{afterextras}{%
1821      \ifcase\bbl@select@type
1822        \bbl@cl{e}%
1823      \fi}%
1824    \begingroup
1825      \let\bbl@ens@include\@empty
1826      \let\bbl@ens@exclude\@empty
1827      \def\bbl@ens@fontenc{\relax}%
1828      \def\bbl@tempb##1{%
1829        \ifx\@empty##1\else\noexpand##1\expandafter\bbl@tempb\fi}%
1830      \edef\bbl@tempa{\bbl@tempb#1\@empty}%
1831      \def\bbl@tempb##1=##2\@@{\@namedef{bbl@ens@##1}{##2}}%
1832      \bbl@foreach\bbl@tempa{\bbl@tempb##1\@@}%
1833      \def\bbl@tempc{\bbl@ensure}%
1834      \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1835        \expandafter{\bbl@ens@include}}%
1836      \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1837        \expandafter{\bbl@ens@exclude}}%
1838      \toks@\expandafter{\bbl@tempc}%
1839      \bbl@exp{%
1840    \endgroup
1841    \def\<bbl@e@#2>{\the\toks@{\bbl@ens@fontenc}}}}}
```

```
1842 \def\bbl@ensure#1#2#3{% 1: include 2: exclude 3: fontenc
1843   \def\bbl@tempb##1{% elt for (excluding) \bbl@captionlist list
1844     \ifx##1\@undefined % 3.32 - Don't assume the macro exists
1845       \edef##1{\noexpand\bbl@nocaption
1846         {\bbl@stripslash##1}{\languagename\bbl@stripslash##1}}%
1847     \fi
1848     \ifx##1\@empty\else
1849       \in@{##1}{#2}%
1850       \ifin@\else
1851         \bbl@ifunset{bbl@ensure@\languagename}%
1852           {\bbl@exp{%
1853             \\\DeclareRobustCommand\<bbl@ensure@\languagename>[1]{%
1854               \\\foreignlanguage{\languagename}%
1855               {\ifx\relax#3\else
1856                 \\\fontencoding{#3}\\\selectfont
1857               \fi
1858               ########1}}}}%
1859           {}%
1860         \toks@\expandafter{##1}%
1861         \edef##1{%
1862           \bbl@csarg\noexpand{ensure@\languagename}%
1863           {\the\toks@}}%
1864       \fi
1865       \expandafter\bbl@tempb
1866     \fi}%
1867   \expandafter\bbl@tempb\bbl@captionlist\today\@empty
1868   \def\bbl@tempa##1{% elt for include list
1869     \ifx##1\@empty\else
1870       \bbl@csarg\in@{ensure@\languagename\expandafter}\expandafter{##1}%
1871       \ifin@\else
1872         \bbl@tempb##1\@empty
1873       \fi
1874       \expandafter\bbl@tempa
1875     \fi}%
1876   \bbl@tempa#1\@empty}
1877 \def\bbl@captionlist{%
1878   \prefacename\refname\abstractname\bibname\chaptername\appendixname
1879   \contentsname\listfigurename\listtablename\indexname\figurename
1880   \tablename\partname\enclname\ccname\headtoname\pagename\seename
1881   \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.

```
1882 \bbl@trace{Macros for setting language files up}
1883 \def\bbl@ldfinit{%
1884   \let\bbl@screset\@empty
1885   \let\BabelStrings\bbl@opt@string
1886   \let\BabelOptions\@empty
1887   \let\BabelLanguages\relax
1888   \ifx\originalTeX\@undefined
1889     \let\originalTeX\@empty
1890   \else
1891     \originalTeX
1892   \fi}
1893 \def\LdfInit#1#2{%
1894   \chardef\atcatcode=\catcode`\@
1895   \catcode`\@=11\relax
1896   \chardef\eqcatcode=\catcode`\=
1897   \catcode`\==12\relax
1898   \expandafter\if\expandafter\@backslashchar
1899                   \expandafter\@car\string#2\@nil
1900     \ifx#2\@undefined\else
1901       \ldf@quit{#1}%
1902     \fi
1903   \else
1904     \expandafter\ifx\csname#2\endcsname\relax\else
1905       \ldf@quit{#1}%
1906     \fi
1907   \fi
1908   \bbl@ldfinit}
```

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

```
1909 \def\ldf@quit#1{%
1910   \expandafter\main@language\expandafter{#1}%
1911   \catcode`\@=\atcatcode \let\atcatcode\relax
1912   \catcode`\==\eqcatcode \let\eqcatcode\relax
1913   \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.

```
1914 \def\bbl@afterldf#1{% TODO. Merge into the next macro? Unused elsewhere
1915   \bbl@afterlang
1916   \let\bbl@afterlang\relax
1917   \let\BabelModifiers\relax
1918   \let\bbl@screset\relax}%
1919 \def\ldf@finish#1{%
1920   \ifx\loadlocalcfg\@undefined\else % For LaTeX 209
1921     \loadlocalcfg{#1}%
1922   \fi
1923   \bbl@afterldf{#1}%
1924   \expandafter\main@language\expandafter{#1}%
1925   \catcode`\@=\atcatcode \let\atcatcode\relax
1926   \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.

```
1927 \@onlypreamble\LdfInit
1928 \@onlypreamble\ldf@quit
1929 \@onlypreamble\ldf@finish
```

\main@language
\bbl@main@language

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

```
1930 \def\main@language#1{%
1931   \def\bbl@main@language{#1}%
1932   \let\languagename\bbl@main@language % TODO. Set localename
1933   \bbl@id@assign
1934   \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.

```
1935 \def\bbl@beforestart{%
1936   \bbl@usehooks{beforestart}{}%
1937   \global\let\bbl@beforestart\relax}
1938 \AtBeginDocument{%
1939   \@nameuse{bbl@beforestart}%
1940   \if@filesw
1941     \providecommand\babel@aux[2]{}%
1942     \immediate\write\@mainaux{%
1943       \string\providecommand\string\babel@aux[2]{}}%
1944     \immediate\write\@mainaux{\string\@nameuse{bbl@beforestart}}%
1945   \fi
1946   \expandafter\selectlanguage\expandafter{\bbl@main@language}%
1947   \ifbbl@single  % must go after the line above.
1948     \renewcommand\selectlanguage[1]{}%
1949     \renewcommand\foreignlanguage[2]{#2}%
1950     \global\let\babel@aux\@gobbletwo  % Also as flag
1951   \fi
1952   \ifcase\bbl@engine\or\pagedir\bodydir\fi} % TODO - a better place
```

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

```
1953 \def\select@language@x#1{%
1954   \ifcase\bbl@select@type
1955     \bbl@ifsamestring\languagename{#1}{}{\select@language{#1}}%
1956   \else
1957     \select@language{#1}%
1958   \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.

```
1959 \bbl@trace{Shorhands}
1960 \def\bbl@add@special#1{% 1:a macro like \", \?, etc.
1961   \bbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
1962   \bbl@ifunset{@sanitize}{}{\bbl@add\@sanitize{\@makeother#1}}%
1963   \ifx\nfss@catcodes\@undefined\else % TODO - same for above
1964     \begingroup
1965       \catcode`#1\active
1966       \nfss@catcodes
```

108

```
1967        \ifnum\catcode`#1=\active
1968          \endgroup
1969          \bbl@add\nfss@catcodes{\@makeother#1}%
1970        \else
1971          \endgroup
1972        \fi
1973   \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.

```
1974 \def\bbl@remove@special#1{%
1975   \begingroup
1976     \def\x##1##2{\ifnum`#1=`##2\noexpand\@empty
1977                  \else\noexpand##1\noexpand##2\fi}%
1978     \def\do{\x\do}%
1979     \def\@makeother{\x\@makeother}%
1980   \edef\x{\endgroup
1981     \def\noexpand\dospecials{\dospecials}%
1982     \expandafter\ifx\csname @sanitize\endcsname\relax\else
1983       \def\noexpand\@sanitize{\@sanitize}%
1984     \fi}%
1985   \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).

```
1986 \def\bbl@active@def#1#2#3#4{%
1987   \@namedef{#3#1}{%
1988     \expandafter\ifx\csname#2@sh@#1@\endcsname\relax
1989       \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
1990     \else
1991       \bbl@afterfi\csname#2@sh@#1@\endcsname
1992     \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.

```
1993   \long\@namedef{#3@arg#1}##1{%
1994     \expandafter\ifx\csname#2@sh@#1@\string##1@\endcsname\relax
1995       \bbl@afterelse\csname#4#1\endcsname##1%
1996     \else
1997       \bbl@afterfi\csname#2@sh@#1@\string##1@\endcsname
1998     \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.

```
1999 \def\initiate@active@char#1{%
2000   \bbl@ifunset{active@char\string#1}%
2001     {\bbl@withactive
2002       {\expandafter\@initiate@active@char\expandafter}#1\string#1#1}%
2003     {}}
```

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).

```
2004 \def\@initiate@active@char#1#2#3{%
2005   \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
2006   \ifx#1\@undefined
2007     \bbl@csarg\edef{oridef@#2}{\let\noexpand#1\noexpand\@undefined}%
2008   \else
2009     \bbl@csarg\let{oridef@@#2}#1%
2010     \bbl@csarg\edef{oridef@#2}{%
2011       \let\noexpand#1%
2012       \expandafter\noexpand\csname bbl@oridef@@#2\endcsname}%
2013   \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*).

```
2014   \ifx#1#3\relax
2015     \expandafter\let\csname normal@char#2\endcsname#3%
2016   \else
2017     \bbl@info{Making #2 an active character}%
2018     \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
2019       \@namedef{normal@char#2}{%
2020         \textormath{#3}{\csname bbl@oridef@@#2\endcsname}}%
2021     \else
2022       \@namedef{normal@char#2}{#3}%
2023     \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).

```
2024       \bbl@restoreactive{#2}%
2025       \AtBeginDocument{%
2026         \catcode`#2\active
2027         \if@filesw
2028           \immediate\write\@mainaux{\catcode`\string#2\active}%
2029         \fi}%
2030       \expandafter\bbl@add@special\csname#2\endcsname
2031       \catcode`#2\active
2032   \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⟩).

```
2033    \let\bbl@tempa\@firstoftwo
2034    \if\string^#2%
2035      \def\bbl@tempa{\noexpand\textormath}%
2036    \else
2037      \ifx\bbl@mathnormal\@undefined\else
2038        \let\bbl@tempa\bbl@mathnormal
2039      \fi
2040    \fi
2041    \expandafter\edef\csname active@char#2\endcsname{%
2042      \bbl@tempa
2043        {\noexpand\if@safe@actives
2044            \noexpand\expandafter
2045            \expandafter\noexpand\csname normal@char#2\endcsname
2046          \noexpand\else
2047            \noexpand\expandafter
2048            \expandafter\noexpand\csname bbl@doactive#2\endcsname
2049          \noexpand\fi}%
2050        {\expandafter\noexpand\csname normal@char#2\endcsname}}%
2051    \bbl@csarg\edef{doactive#2}{%
2052      \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$\langle char \rangle$ is *one* control sequence!).

```
2053    \bbl@csarg\edef{active@#2}{%
2054      \noexpand\active@prefix\noexpand#1%
2055      \expandafter\noexpand\csname active@char#2\endcsname}%
2056    \bbl@csarg\edef{normal@#2}{%
2057      \noexpand\active@prefix\noexpand#1%
2058      \expandafter\noexpand\csname normal@char#2\endcsname}%
2059    \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.

```
2060    \bbl@active@def#2\user@group{user@active}{language@active}%
2061    \bbl@active@def#2\language@group{language@active}{system@active}%
2062    \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.

```
2063    \expandafter\edef\csname\user@group @sh@#2@@\endcsname
2064      {\expandafter\noexpand\csname normal@char#2\endcsname}%
2065    \expandafter\edef\csname\user@group @sh@#2@\string\protect@\endcsname
2066      {\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.

```
2067    \if\string'#2%
2068      \let\prim@s\bbl@prim@s
2069      \let\active@math@prime#1%
```

```
2070    \fi
2071    \bbl@usehooks{initiateactive}{{#1}{#2}{#3}}}
```

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

```
2072 ⟨⟨*More package options⟩⟩ ≡
2073 \DeclareOption{math=active}{}
2074 \DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
2075 ⟨⟨/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`.

```
2076 \@ifpackagewith{babel}{KeepShorthandsActive}%
2077   {\let\bbl@restoreactive\@gobble}%
2078   {\def\bbl@restoreactive#1{%
2079     \bbl@exp{%
2080       \\\AfterBabelLanguage\\\CurrentOption
2081         {\catcode`#1=\the\catcode`#1\relax}%
2082       \\\AtEndOfPackage
2083         {\catcode`#1=\the\catcode`#1\relax}}}%
2084   \AtEndOfPackage{\let\bbl@restoreactive\@gobble}}
```

\bbl@sh@select   This command helps the shorthand supporting macros to select how to proceed. Note that this macro needs to be expandable as do all the shorthand macros in order for them to work in expansion-only environments such as the argument of `\hyphenation`.
This macro expects the name of a group of shorthands in its first argument and a shorthand character in its second argument. It will expand to either `\bbl@firstcs` or `\bbl@scndcs`. Hence two more arguments need to follow it.

```
2085 \def\bbl@sh@select#1#2{%
2086   \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
2087     \bbl@afterelse\bbl@scndcs
2088   \else
2089     \bbl@afterfi\csname#1@sh@#2@sel\endcsname
2090   \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.

```
2091 \begingroup
2092 \bbl@ifunset{ifincsname}% TODO. Ugly. Correct?
2093   {\gdef\active@prefix#1{%
2094     \ifx\protect\@typeset@protect
2095     \else
2096       \ifx\protect\@unexpandable@protect
2097         \noexpand#1%
2098       \else
2099         \protect#1%
2100       \fi
2101       \expandafter\@gobble
2102     \fi}}
2103   {\gdef\active@prefix#1{%
2104     \ifincsname
2105       \string#1%
2106       \expandafter\@gobble
2107     \else
2108       \ifx\protect\@typeset@protect
2109       \else
```

```
2110        \ifx\protect\@unexpandable@protect
2111          \noexpand#1%
2112        \else
2113          \protect#1%
2114        \fi
2115        \expandafter\expandafter\expandafter\@gobble
2116      \fi
2117    \fi}}
2118 \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*⟩.

```
2119 \newif\if@safe@actives
2120 \@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.

```
2121 \def\bbl@restore@actives{\if@safe@actives\@safe@activesfalse\fi}
```

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

```
2122 \chardef\bbl@activated\z@
2123 \def\bbl@activate#1{%
2124   \chardef\bbl@activated\@ne
2125   \bbl@withactive{\expandafter\let\expandafter}#1%
2126     \csname bbl@active@\string#1\endcsname}
2127 \def\bbl@deactivate#1{%
2128   \chardef\bbl@activated\tw@
2129   \bbl@withactive{\expandafter\let\expandafter}#1%
2130     \csname bbl@normal@\string#1\endcsname}
```

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

```
2131 \def\bbl@firstcs#1#2{\csname#1\endcsname}
2132 \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.

```
2133 \def\babel@texpdf#1#2#3#4{%
2134   \ifx\texorpdfstring\@undefined
2135     \textormath{#1}{#3}%
2136   \else
2137     \texorpdfstring{\textormath{#1}{#3}}{#2}%
2138   % \texorpdfstring{\textormath{#1}{#3}}{\textormath{#2}{#4}}%
2139   \fi}
2140 %
```

```
2141 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
2142 \def\@decl@short#1#2#3\@nil#4{%
2143   \def\bbl@tempa{#3}%
2144   \ifx\bbl@tempa\@empty
2145     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
2146     \bbl@ifunset{#1@sh@\string#2@}{}%
2147       {\def\bbl@tempa{#4}%
2148        \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbl@tempa
2149        \else
2150          \bbl@info
2151            {Redefining #1 shorthand \string#2\\%
2152             in language \CurrentOption}%
2153        \fi}%
2154     \@namedef{#1@sh@\string#2@}{#4}%
2155   \else
2156     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs
2157     \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%
2158       {\def\bbl@tempa{#4}%
2159        \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
2160        \else
2161          \bbl@info
2162            {Redefining #1 shorthand \string#2\string#3\\%
2163             in language \CurrentOption}%
2164        \fi}%
2165     \@namedef{#1@sh@\string#2@\string#3@}{#4}%
2166   \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.

```
2167 \def\textormath{%
2168   \ifmmode
2169     \expandafter\@secondoftwo
2170   \else
2171     \expandafter\@firstoftwo
2172   \fi}
```

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

```
2173 \def\user@group{user}
2174 \def\language@group{english} % TODO. I don't like defaults
2175 \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.

```
2176 \def\useshorthands{%
2177   \@ifstar\bbl@usesh@s{\bbl@usesh@x{}}}
2178 \def\bbl@usesh@s#1{%
2179   \bbl@usesh@x
2180     {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bbl@activate{#1}}}%
2181     {#1}}
2182 \def\bbl@usesh@x#1#2{%
2183   \bbl@ifshorthand{#2}%
2184     {\def\user@group{user}%
2185      \initiate@active@char{#2}%
2186      #1%
2187      \bbl@activate{#2}}%
```

```
2188    {\bbl@error
2189      {Cannot declare a shorthand turned off (\string#2)}
2190      {Sorry, but you cannot use shorthands which have been\\%
2191       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.

```
2192 \def\user@language@group{user@\language@group}
2193 \def\bbl@set@user@generic#1#2{%
2194   \bbl@ifunset{user@generic@active#1}%
2195     {\bbl@active@def#1\user@language@group{user@active}{user@generic@active}%
2196      \bbl@active@def#1\user@group{user@generic@active}{language@active}%
2197      \expandafter\edef\csname#2@sh@#1@@\endcsname{%
2198        \expandafter\noexpand\csname normal@char#1\endcsname}%
2199      \expandafter\edef\csname#2@sh@#1@\string\protect@\endcsname{%
2200        \expandafter\noexpand\csname user@active#1\endcsname}}%
2201     \@empty}
2202 \newcommand\defineshorthand[3][user]{%
2203   \edef\bbl@tempa{\zap@space#1 \@empty}%
2204   \bbl@for\bbl@tempb\bbl@tempa{%
2205     \if*\expandafter\@car\bbl@tempb\@nil
2206       \edef\bbl@tempb{user@\expandafter\@gobble\bbl@tempb}%
2207       \@expandtwoargs
2208         \bbl@set@user@generic{\expandafter\string\@car#2\@nil}\bbl@tempb
2209     \fi
2210     \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].

```
2211 \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".

```
2212 \def\aliasshorthand#1#2{%
2213   \bbl@ifshorthand{#2}%
2214     {\expandafter\ifx\csname active@char\string#2\endcsname\relax
2215       \ifx\document\@notprerr
2216         \@notshorthand{#2}%
2217       \else
2218         \initiate@active@char{#2}%
2219         \expandafter\let\csname active@char\string#2\expandafter\endcsname
2220           \csname active@char\string#1\endcsname
2221         \expandafter\let\csname normal@char\string#2\expandafter\endcsname
2222           \csname normal@char\string#1\endcsname
2223         \bbl@activate{#2}%
2224       \fi
2225     \fi}%
2226     {\bbl@error
2227       {Cannot declare a shorthand turned off (\string#2)}
2228       {Sorry, but you cannot use shorthands which have been\\%
2229       turned off in the package options}}}
```

\@notshorthand

```
2230 \def\@notshorthand#1{%
2231   \bbl@error{%
2232     The character `\string #1' should be made a shorthand character;\\%
2233     add the command \string\useshorthands\string{#1\string} to
2234     the preamble.\\%
2235     I will ignore your instruction}%
2236     {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.

```
2237 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
2238 \DeclareRobustCommand*\shorthandoff{%
2239   \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
2240 \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.

```
2241 \def\bbl@switch@sh#1#2{%
2242   \ifx#2\@nnil\else
2243     \bbl@ifunset{bbl@active@\string#2}%
2244       {\bbl@error
2245         {I cannot switch `\string#2' on or off--not a shorthand}%
2246         {This character is not a shorthand. Maybe you made\\%
2247          a typing mistake? I will ignore your instruction.}}%
2248       {\ifcase#1%   off, on, off*
2249         \catcode`#212\relax
2250       \or
2251         \catcode`#2\active
2252         \bbl@ifunset{bbl@shdef@\string#2}%
2253           {}%
2254           {\bbl@withactive{\expandafter\let\expandafter}#2%
2255             \csname bbl@shdef@\string#2\endcsname
2256            \bbl@csarg\let{shdef@\string#2}\relax}%
2257         \ifcase\bbl@activated\or
2258           \bbl@activate{#2}%
2259         \else
2260           \bbl@deactivate{#2}%
2261         \fi
2262       \or
2263         \bbl@ifunset{bbl@shdef@\string#2}%
2264           {\bbl@withactive{\bbl@csarg\let{shdef@\string#2}}#2}%
2265           {}%
2266         \csname bbl@oricat@\string#2\endcsname
2267         \csname bbl@oridef@\string#2\endcsname
2268       \fi}%
2269     \bbl@afterfi\bbl@switch@sh#1%
2270   \fi}
```

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

```
2271 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
2272 \def\bbl@putsh#1{%
2273   \bbl@ifunset{bbl@active@\string#1}%
2274     {\bbl@putsh@i#1\@empty\@nnil}%
```

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

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

```
2296 \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.

```
2297 \def\bbl@prim@s{%
2298   \prime\futurelet\@let@token\bbl@pr@m@s}
2299 \def\bbl@if@primes#1#2{%
2300   \ifx#1\@let@token
2301     \expandafter\@firstoftwo
2302   \else\ifx#2\@let@token
2303     \bbl@afterelse\expandafter\@firstoftwo
2304   \else
2305     \bbl@afterfi\expandafter\@secondoftwo
2306   \fi\fi}
2307 \begingroup
2308   \catcode`\^=7  \catcode`\*=\active  \lccode`\*=`\^
2309   \catcode`\'=12 \catcode`\"=\active  \lccode`\"=`\'
2310   \lowercase{%
2311   \gdef\bbl@pr@m@s{%
2312     \bbl@if@primes"'%
2313       \pr@@@s
2314       {\bbl@if@primes*^\pr@@@t\egroup}}}
2315 \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).

```
2316 \initiate@active@char{~}
2317 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
2318 \bbl@activate{~}
```

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

```
2319 \expandafter\def\csname OT1dqpos\endcsname{127}
2320 \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

```
2321 \ifx\f@encoding\@undefined
2322   \def\f@encoding{OT1}
2323 \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.

```
2324 \bbl@trace{Language attributes}
2325 \newcommand\languageattribute[2]{%
2326   \def\bbl@tempc{#1}%
2327   \bbl@fixname\bbl@tempc
2328   \bbl@iflanguage\bbl@tempc{%
2329     \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.

```
2330       \ifx\bbl@known@attribs\@undefined
2331         \in@false
2332       \else
2333         \bbl@xin@{,\bbl@tempc-##1,}{,\bbl@known@attribs,}%
2334       \fi
2335       \ifin@
2336         \bbl@warning{%
2337           You have more than once selected the attribute '##1'\\%
2338           for language #1. Reported}%
2339       \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.

```
2340         \bbl@exp{%
2341           \\\bbl@add@list\\\bbl@known@attribs{\bbl@tempc-##1}}%
2342         \edef\bbl@tempa{\bbl@tempc-##1}%
2343         \expandafter\bbl@ifknown@ttrib\expandafter{\bbl@tempa}\bbl@attributes%
2344         {\csname\bbl@tempc @attr@##1\endcsname}%
2345         {\@attrerr{\bbl@tempc}{##1}}%
2346     \fi}}}
2347 \@onlypreamble\languageattribute
```

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

```
2348 \newcommand*{\@attrerr}[2]{%
2349   \bbl@error
2350     {The attribute #2 is unknown for language #1.}%
2351     {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}.

```
2352 \def\bbl@declare@ttribute#1#2#3{%
2353   \bbl@xin@{,#2,}{,\BabelModifiers,}%
2354   \ifin@
2355     \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
2356   \fi
2357   \bbl@add@list\bbl@attributes{#1-#2}%
2358   \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.

```
2359 \def\bbl@ifattributeset#1#2#3#4{%
2360   \ifx\bbl@known@attribs\@undefined
2361     \in@false
2362   \else
2363     \bbl@xin@{,#1-#2,}{,\bbl@known@attribs,}%
2364   \fi
2365   \ifin@
2366     \bbl@afterelse#3%
2367   \else
2368     \bbl@afterfi#4%
2369   \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.

```
2370 \def\bbl@ifknown@ttrib#1#2{%
2371   \let\bbl@tempa\@secondoftwo
2372   \bbl@loopx\bbl@tempb{#2}{%
2373     \expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{,#1,}%
2374     \ifin@
2375       \let\bbl@tempa\@firstoftwo
2376     \else
2377     \fi}%
2378   \bbl@tempa}
```

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

```
2379 \def\bbl@clear@ttribs{%
2380   \ifx\bbl@attributes\@undefined\else
2381     \bbl@loopx\bbl@tempa{\bbl@attributes}{%
2382       \expandafter\bbl@clear@ttrib\bbl@tempa.
2383       }%
2384     \let\bbl@attributes\@undefined
2385   \fi}
2386 \def\bbl@clear@ttrib#1-#2.{%
2387   \expandafter\let\csname#1@attr@#2\endcsname\@undefined}
2388 \AtBeginDocument{\bbl@clear@ttribs}
```

119

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
```
2389 \bbl@trace{Macros for saving definitions}
2390 \def\babel@beginsave{\babel@savecnt\z@}
```

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

```
2391 \newcount\babel@savecnt
2392 \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.

```
2393 \def\babel@save#1{%
2394   \expandafter\let\csname babel@\number\babel@savecnt\endcsname#1\relax
2395   \toks@\expandafter{\originalTeX\let#1=}%
2396   \bbl@exp{%
2397     \def\\\originalTeX{\the\toks@\<babel@\number\babel@savecnt>\relax}}%
2398   \advance\babel@savecnt\@ne}
2399 \def\babel@savevariable#1{%
2400   \toks@\expandafter{\originalTeX #1=}%
2401   \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.

```
2402 \def\bbl@frenchspacing{%
2403   \ifnum\the\sfcode`\.=\@m
2404     \let\bbl@nonfrenchspacing\relax
2405   \else
2406     \frenchspacing
2407     \let\bbl@nonfrenchspacing\nonfrenchspacing
2408   \fi}
2409 \let\bbl@nonfrenchspacing\nonfrenchspacing
2410 \let\bbl@elt\relax
2411 \edef\bbl@fs@chars{%
2412   \bbl@elt{\string.}\@m{3000}\bbl@elt{\string?}\@m{3000}%
2413   \bbl@elt{\string!}\@m{3000}\bbl@elt{\string:}\@m{2000}%
2414   \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.

---

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

```
2415 \bbl@trace{Short tags}
2416 \def\babeltags#1{%
2417   \edef\bbl@tempa{\zap@space#1 \@empty}%
2418   \def\bbl@tempb##1=##2\@@{%
2419     \edef\bbl@tempc{%
2420       \noexpand\newcommand
2421       \expandafter\noexpand\csname ##1\endcsname{%
2422         \noexpand\protect
2423         \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}%
2424       \noexpand\newcommand
2425       \expandafter\noexpand\csname text##1\endcsname{%
2426         \noexpand\foreignlanguage{##2}}}%
2427     \bbl@tempc}%
2428   \bbl@for\bbl@tempa\bbl@tempa{%
2429     \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.

```
2430 \bbl@trace{Hyphens}
2431 \@onlypreamble\babelhyphenation
2432 \AtEndOfPackage{%
2433   \newcommand\babelhyphenation[2][\@empty]{%
2434     \ifx\bbl@hyphenation@\relax
2435       \let\bbl@hyphenation@\@empty
2436     \fi
2437     \ifx\bbl@hyphlist\@empty\else
2438       \bbl@warning{%
2439         You must not intermingle \string\selectlanguage\space and\\%
2440         \string\babelhyphenation\space or some exceptions will not\\%
2441         be taken into account. Reported}%
2442     \fi
2443     \ifx\@empty#1%
2444       \protected@edef\bbl@hyphenation@{\bbl@hyphenation@\space#2}%
2445     \else
2446       \bbl@vforeach{#1}{%
2447         \def\bbl@tempa{##1}%
2448         \bbl@fixname\bbl@tempa
2449         \bbl@iflanguage\bbl@tempa{%
2450           \bbl@csarg\protected@edef{hyphenation@\bbl@tempa}{%
2451             \bbl@ifunset{bbl@hyphenation@\bbl@tempa}%
2452               {}%
2453               {\csname bbl@hyphenation@\bbl@tempa\endcsname\space}%
2454             #2}}}%
2455     \fi}}
```

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

```
2456 \def\bbl@allowhyphens{\ifvmode\else\nobreak\hskip\z@skip\fi}
2457 \def\bbl@t@one{T1}
2458 \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`.

---

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

```
2459 \newcommand\babelnullhyphen{\char\hyphenchar\font}
2460 \def\babelhyphen{\active@prefix\babelhyphen\bbl@hyphen}
2461 \def\bbl@hyphen{%
2462   \@ifstar{\bbl@hyphen@i @}{\bbl@hyphen@i\@empty}}
2463 \def\bbl@hyphen@i#1#2{%
2464   \bbl@ifunset{bbl@hy@#1#2\@empty}%
2465     {\csname bbl@#1usehyphen\endcsname{\discretionary{#2}{}{#2}}}%
2466     {\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.

```
2467 \def\bbl@usehyphen#1{%
2468   \leavevmode
2469   \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
2470   \nobreak\hskip\z@skip}
2471 \def\bbl@@usehyphen#1{%
2472   \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}
```

The following macro inserts the hyphen char.

```
2473 \def\bbl@hyphenchar{%
2474   \ifnum\hyphenchar\font=\m@ne
2475     \babelnullhyphen
2476   \else
2477     \char\hyphenchar\font
2478   \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.

```
2479 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}{}}}
2480 \def\bbl@hy@@soft{\bbl@@usehyphen{\discretionary{\bbl@hyphenchar}{}{}}}
2481 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
2482 \def\bbl@hy@@hard{\bbl@@usehyphen\bbl@hyphenchar}
2483 \def\bbl@hy@nobreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}}
2484 \def\bbl@hy@@nobreak{\mbox{\bbl@hyphenchar}}
2485 \def\bbl@hy@repeat{%
2486   \bbl@usehyphen{%
2487     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
2488 \def\bbl@hy@@repeat{%
2489   \bbl@@usehyphen{%
2490     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
2491 \def\bbl@hy@empty{\hskip\z@skip}
2492 \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.

```
2493 \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.

```
2494 \bbl@trace{Multiencoding strings}
2495 \def\bbl@toglobal#1{\global\let#1#1}
2496 \def\bbl@recatcode#1{% TODO. Used only once?
2497   \@tempcnta="7F
2498   \def\bbl@tempa{%
2499     \ifnum\@tempcnta>"FF\else
2500       \catcode\@tempcnta=#1\relax
2501       \advance\@tempcnta\@ne
2502       \expandafter\bbl@tempa
2503     \fi}%
2504   \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).

```
2505 \@ifpackagewith{babel}{nocase}%
2506   {\let\bbl@patchuclc\relax}%
2507   {\def\bbl@patchuclc{%
2508     \global\let\bbl@patchuclc\relax
2509     \g@addto@macro\@uclclist{\reserved@b{\reserved@b\bbl@uclc}}%
2510     \gdef\bbl@uclc##1{%
2511       \let\bbl@encoded\bbl@encoded@uclc
2512       \bbl@ifunset{\languagename @bbl@uclc}% and resumes it
2513         {##1}%
2514         {\let\bbl@tempa##1\relax % Used by LANG@bbl@uclc
2515          \csname\languagename @bbl@uclc\endcsname}%
2516       {\bbl@tolower\@empty}{\bbl@toupper\@empty}}%
2517     \gdef\bbl@tolower{\csname\languagename @bbl@lc\endcsname}%
2518     \gdef\bbl@toupper{\csname\languagename @bbl@uc\endcsname}}}
```

```
2519 ⟨⟨*More package options⟩⟩ ≡
2520 \DeclareOption{nocase}{}
2521 ⟨⟨/More package options⟩⟩
```

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

```
2522 ⟨⟨*More package options⟩⟩ ≡
2523 \let\bbl@opt@strings\@nnil % accept strings=value
2524 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
2525 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
2526 \def\BabelStringsDefault{generic}
2527 ⟨⟨/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.

```
2528 \@onlypreamble\StartBabelCommands
2529 \def\StartBabelCommands{%
2530   \begingroup
```

```
2531    \bbl@recatcode{11}%
2532    ⟨⟨Macros local to BabelCommands⟩⟩
2533    \def\bbl@provstring##1##2{%
2534      \providecommand##1{##2}%
2535      \bbl@toglobal##1}%
2536    \global\let\bbl@scafter\@empty
2537    \let\StartBabelCommands\bbl@startcmds
2538    \ifx\BabelLanguages\relax
2539        \let\BabelLanguages\CurrentOption
2540    \fi
2541    \begingroup
2542    \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
2543    \StartBabelCommands}
2544 \def\bbl@startcmds{%
2545    \ifx\bbl@screset\@nnil\else
2546      \bbl@usehooks{stopcommands}{}%
2547    \fi
2548    \endgroup
2549    \begingroup
2550    \@ifstar
2551      {\ifx\bbl@opt@strings\@nnil
2552          \let\bbl@opt@strings\BabelStringsDefault
2553        \fi
2554        \bbl@startcmds@i}%
2555      \bbl@startcmds@i}
2556 \def\bbl@startcmds@i#1#2{%
2557    \edef\bbl@L{\zap@space#1 \@empty}%
2558    \edef\bbl@G{\zap@space#2 \@empty}%
2559    \bbl@startcmds@ii}
2560 \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.

```
2561 \newcommand\bbl@startcmds@ii[1][\@empty]{%
2562    \let\SetString\@gobbletwo
2563    \let\bbl@stringdef\@gobbletwo
2564    \let\AfterBabelCommands\@gobble
2565    \ifx\@empty#1%
2566      \def\bbl@sc@label{generic}%
2567      \def\bbl@encstring##1##2{%
2568        \ProvideTextCommandDefault##1{##2}%
2569        \bbl@toglobal##1%
2570        \expandafter\bbl@toglobal\csname\string?\string##1\endcsname}%
2571      \let\bbl@sctest\in@true
2572    \else
2573      \let\bbl@sc@charset\space % <- zapped below
2574      \let\bbl@sc@fontenc\space % <-    "       "
2575      \def\bbl@tempa##1=##2\@nil{%
2576        \bbl@csarg\edef{sc@\zap@space##1 \@empty}{##2 }}%
2577      \bbl@vforeach{label=#1}{\bbl@tempa##1\@nil}%
2578      \def\bbl@tempa##1 ##2{% space -> comma
2579        ##1%
```

```
2580        \ifx\@empty##2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa##2\fi}%
2581     \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%
2582     \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%
2583     \edef\bbl@sc@charset{\expandafter\zap@space\bbl@sc@charset\@empty}%
2584     \def\bbl@encstring##1##2{%
2585        \bbl@foreach\bbl@sc@fontenc{%
2586           \bbl@ifunset{T@####1}%
2587              {}%
2588              {\ProvideTextCommand##1{####1}{##2}%
2589               \bbl@toglobal##1%
2590               \expandafter
2591               \bbl@toglobal\csname####1\string##1\endcsname}}}%
2592     \def\bbl@sctest{%
2593        \bbl@xin@{,\bbl@opt@strings,}{,\bbl@sc@label,\bbl@sc@fontenc,}}%
2594  \fi
2595  \ifx\bbl@opt@strings\@nnil        % ie, no strings key -> defaults
2596  \else\ifx\bbl@opt@strings\relax   % ie, strings=encoded
2597     \let\AfterBabelCommands\bbl@aftercmds
2598     \let\SetString\bbl@setstring
2599     \let\bbl@stringdef\bbl@encstring
2600  \else        % ie, strings=value
2601     \bbl@sctest
2602     \ifin@
2603        \let\AfterBabelCommands\bbl@aftercmds
2604        \let\SetString\bbl@setstring
2605        \let\bbl@stringdef\bbl@provstring
2606  \fi\fi\fi
2607  \bbl@scswitch
2608  \ifx\bbl@G\@empty
2609     \def\SetString##1##2{%
2610        \bbl@error{Missing group for string \string##1}%
2611           {You must assign strings to some category, typically\\%
2612            captions or extras, but you set none}}%
2613  \fi
2614  \ifx\@empty#1%
2615     \bbl@usehooks{defaultcommands}{}%
2616  \else
2617     \@expandtwoargs
2618     \bbl@usehooks{encodedcommands}{{\bbl@sc@charset}{\bbl@sc@fontenc}}%
2619  \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) .

```
2620  \def\bbl@forlang#1#2{%
2621     \bbl@for#1\bbl@L{%
2622        \bbl@xin@{,#1,}{,\BabelLanguages,}%
2623        \ifin@#2\relax\fi}}
2624  \def\bbl@scswitch{%
2625     \bbl@forlang\bbl@tempa{%
2626        \ifx\bbl@G\@empty\else
2627           \ifx\SetString\@gobbletwo\else
2628              \edef\bbl@GL{\bbl@G\bbl@tempa}%
2629              \bbl@xin@{,\bbl@GL,}{,\bbl@screset,}%
```

```
2630        \ifin@\else
2631          \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
2632          \xdef\bbl@screset{\bbl@screset,\bbl@GL}%
2633        \fi
2634      \fi
2635    \fi}}
2636 \AtEndOfPackage{%
2637   \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{}{#2}}}%
2638   \let\bbl@scswitch\relax}
2639 \@onlypreamble\EndBabelCommands
2640 \def\EndBabelCommands{%
2641   \bbl@usehooks{stopcommands}{}%
2642   \endgroup
2643   \endgroup
2644   \bbl@scafter}
2645 \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.

```
2646 \def\bbl@setstring#1#2{% eg, \prefacename{<string>}
2647   \bbl@forlang\bbl@tempa{%
2648     \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
2649     \bbl@ifunset{\bbl@LC}% eg, \germanchaptername
2650       {\bbl@exp{%
2651         \global\\\bbl@add\<\bbl@G\bbl@tempa>{\\\bbl@scset\\#1\<\bbl@LC>}}}%
2652       {}%
2653     \def\BabelString{#2}%
2654     \bbl@usehooks{stringprocess}{}%
2655     \expandafter\bbl@stringdef
2656       \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.

```
2657 \ifx\bbl@opt@strings\relax
2658   \def\bbl@scset#1#2{\def#1{\bbl@encoded#2}}
2659   \bbl@patchuclc
2660   \let\bbl@encoded\relax
2661   \def\bbl@encoded@uclc#1{%
2662     \@inmathwarn#1%
2663     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
2664       \expandafter\ifx\csname ?\string#1\endcsname\relax
2665         \TextSymbolUnavailable#1%
2666       \else
2667         \csname ?\string#1\endcsname
2668       \fi
2669     \else
2670       \csname\cf@encoding\string#1\endcsname
2671     \fi}
2672 \else
2673   \def\bbl@scset#1#2{\def#1{#2}}
2674 \fi
```

126

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.

```
2675 ⟨∗Macros local to BabelCommands⟩ ≡
2676 \def\SetStringLoop##1##2{%
2677     \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
2678     \count@\z@
2679     \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
2680         \advance\count@\@ne
2681         \toks@\expandafter{\bbl@tempa}%
2682         \bbl@exp{%
2683             \\\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
2684             \count@=\the\count@\relax}}}%
2685 ⟨/Macros local to BabelCommands⟩
```

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

```
2686 \def\bbl@aftercmds#1{%
2687     \toks@\expandafter{\bbl@scafter#1}%
2688     \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.

```
2689 ⟨∗Macros local to BabelCommands⟩ ≡
2690     \newcommand\SetCase[3][]{%
2691         \bbl@patchuclc
2692         \bbl@forlang\bbl@tempa{%
2693             \expandafter\bbl@encstring
2694                 \csname\bbl@tempa @bbl@uclc\endcsname{\bbl@tempa##1}%
2695             \expandafter\bbl@encstring
2696                 \csname\bbl@tempa @bbl@uc\endcsname{##2}%
2697             \expandafter\bbl@encstring
2698                 \csname\bbl@tempa @bbl@lc\endcsname{##3}}}%
2699 ⟨/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.

```
2700 ⟨∗Macros local to BabelCommands⟩ ≡
2701     \newcommand\SetHyphenMap[1]{%
2702         \bbl@forlang\bbl@tempa{%
2703             \expandafter\bbl@stringdef
2704                 \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
2705 ⟨/Macros local to BabelCommands⟩
```

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

```
2706 \newcommand\BabelLower[2]{% one to one.
2707     \ifnum\lccode#1=#2\else
2708         \babel@savevariable{\lccode#1}%
2709         \lccode#1=#2\relax
2710     \fi}
2711 \newcommand\BabelLowerMM[4]{% many-to-many
2712     \@tempcnta=#1\relax
2713     \@tempcntb=#4\relax
2714     \def\bbl@tempa{%
2715         \ifnum\@tempcnta>#2\else
2716             \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
```

```
2717        \advance\@tempcnta#3\relax
2718        \advance\@tempcntb#3\relax
2719        \expandafter\bbl@tempa
2720      \fi}%
2721    \bbl@tempa}
2722 \newcommand\BabelLowerMO[4]{% many-to-one
2723    \@tempcnta=#1\relax
2724    \def\bbl@tempa{%
2725      \ifnum\@tempcnta>#2\else
2726        \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
2727        \advance\@tempcnta#3
2728        \expandafter\bbl@tempa
2729      \fi}%
2730    \bbl@tempa}
```

The following package options control the behavior of hyphenation mapping.

```
2731 ⟨*More package options⟩ ≡
2732 \DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\z@}
2733 \DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap\@ne}
2734 \DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\tw@}
2735 \DeclareOption{hyphenmap=other}{\chardef\bbl@opt@hyphenmap\thr@@}
2736 \DeclareOption{hyphenmap=other*}{\chardef\bbl@opt@hyphenmap4\relax}
2737 ⟨/More package options⟩
```

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

```
2738 \AtEndOfPackage{%
2739    \ifx\bbl@opt@hyphenmap\@undefined
2740      \bbl@xin@{,}{\bbl@language@opts}%
2741      \chardef\bbl@opt@hyphenmap\ifin@4\else\@ne\fi
2742    \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.

```
2743 \newcommand\setlocalecaption{% TODO. Catch typos. What about ensure?
2744    \@ifstar\bbl@setcaption@s\bbl@setcaption@x}
2745 \def\bbl@setcaption@x#1#2#3{% language caption-name string
2746    \bbl@trim@def\bbl@tempa{#2}%
2747    \bbl@xin@{.template}{\bbl@tempa}%
2748    \ifin@
2749      \bbl@ini@captions@template{#3}{#1}%
2750    \else
2751      \edef\bbl@tempd{%
2752        \expandafter\expandafter\expandafter
2753        \strip@prefix\expandafter\meaning\csname captions#1\endcsname}%
2754      \bbl@xin@
2755        {\expandafter\string\csname #2name\endcsname}%
2756        {\bbl@tempd}%
2757      \ifin@ % Renew caption
2758        \bbl@xin@{\string\bbl@scset}{\bbl@tempd}%
2759        \ifin@
2760          \bbl@exp{%
2761            \\\bbl@ifsamestring{\bbl@tempa}{\languagename}%
2762              {\\\bbl@scset\<#2name>\<#1#2name>}%
2763              {}}%
2764        \else % Old way converts to new way
2765          \bbl@ifunset{#1#2name}%
2766            {\bbl@exp{%
2767              \\\bbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}}%
```

128

```
2768            \\\bbl@ifsamestring{\bbl@tempa}{\languagename}%
2769              {\def\<#2name>{\<#1#2name>}}%
2770              {}}}%
2771          {}%
2772        \fi
2773      \else
2774        \bbl@xin@{\string\bbl@scset}{\bbl@tempd}% New
2775        \ifin@ % New way
2776          \bbl@exp{%
2777            \\\bbl@add\<captions#1>{\\\bbl@scset\<#2name>\<#1#2name>}%
2778            \\\bbl@ifsamestring{\bbl@tempa}{\languagename}%
2779              {\\\bbl@scset\<#2name>\<#1#2name>}%
2780              {}}%
2781        \else  % Old way, but defined in the new way
2782          \bbl@exp{%
2783            \\\bbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}%
2784            \\\bbl@ifsamestring{\bbl@tempa}{\languagename}%
2785              {\def\<#2name>{\<#1#2name>}}%
2786              {}}%
2787        \fi%
2788      \fi
2789      \@namedef{#1#2name}{#3}%
2790      \toks@\expandafter{\bbl@captionslist}%
2791      \bbl@exp{\\\in@{\<#2name>}{\the\toks@}}%
2792      \ifin@\else
2793        \bbl@exp{\\\bbl@add\\\bbl@captionslist{\<#2name>}}%
2794        \bbl@toglobal\bbl@captionslist
2795      \fi
2796    \fi}
2797 % \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.

```
2798 \bbl@trace{Macros related to glyphs}
2799 \def\set@low@box#1{\setbox\tw@\hbox{,}\setbox\z@\hbox{#1}%
2800    \dimen\z@\ht\z@ \advance\dimen\z@ -\ht\tw@%
2801    \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.

```
2802 \def\save@sf@q#1{\leavevmode
2803    \begingroup
2804    \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
2805    \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.

```
2806 \ProvideTextCommand{\quotedblbase}{OT1}{%
2807    \save@sf@q{\set@low@box{\textquotedblright\/}%
```

```
2808        \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.

```
2809 \ProvideTextCommandDefault{\quotedblbase}{%
2810    \UseTextSymbol{OT1}{\quotedblbase}}
```

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

```
2811 \ProvideTextCommand{\quotesinglbase}{OT1}{%
2812    \save@sf@q{\set@low@box{\textquoteright\/}%
2813        \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.

```
2814 \ProvideTextCommandDefault{\quotesinglbase}{%
2815    \UseTextSymbol{OT1}{\quotesinglbase}}
```

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

```
2816 \ProvideTextCommand{\guillemetleft}{OT1}{%
2817    \ifmmode
2818        \ll
2819    \else
2820        \save@sf@q{\nobreak
2821            \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2822    \fi}
2823 \ProvideTextCommand{\guillemetright}{OT1}{%
2824    \ifmmode
2825        \gg
2826    \else
2827        \save@sf@q{\nobreak
2828            \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2829    \fi}
2830 \ProvideTextCommand{\guillemotleft}{OT1}{%
2831    \ifmmode
2832        \ll
2833    \else
2834        \save@sf@q{\nobreak
2835            \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2836    \fi}
2837 \ProvideTextCommand{\guillemotright}{OT1}{%
2838    \ifmmode
2839        \gg
2840    \else
2841        \save@sf@q{\nobreak
2842            \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2843    \fi}
```

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

```
2844 \ProvideTextCommandDefault{\guillemetleft}{%
2845    \UseTextSymbol{OT1}{\guillemetleft}}
2846 \ProvideTextCommandDefault{\guillemetright}{%
2847    \UseTextSymbol{OT1}{\guillemetright}}
2848 \ProvideTextCommandDefault{\guillemotleft}{%
2849    \UseTextSymbol{OT1}{\guillemotleft}}
2850 \ProvideTextCommandDefault{\guillemotright}{%
2851    \UseTextSymbol{OT1}{\guillemotright}}
```

\guilsinglleft   The single guillemets are not available in OT1 encoding. They are faked.
\guilsinglright
```
2852 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2853    \ifmmode
```

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

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

```
2866 \ProvideTextCommandDefault{\guilsinglleft}{%
2867    \UseTextSymbol{OT1}{\guilsinglleft}}
2868 \ProvideTextCommandDefault{\guilsinglright}{%
2869    \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.

```
2870 \DeclareTextCommand{\ij}{OT1}{%
2871    i\kern-0.02em\bbl@allowhyphens j}
2872 \DeclareTextCommand{\IJ}{OT1}{%
2873    I\kern-0.02em\bbl@allowhyphens J}
2874 \DeclareTextCommand{\ij}{T1}{\char188}
2875 \DeclareTextCommand{\IJ}{T1}{\char156}
```

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

```
2876 \ProvideTextCommandDefault{\ij}{%
2877    \UseTextSymbol{OT1}{\ij}}
2878 \ProvideTextCommandDefault{\IJ}{%
2879    \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).

```
2880 \def\crrtic@{\hrule height0.1ex width0.3em}
2881 \def\crttic@{\hrule height0.1ex width0.33em}
2882 \def\ddj@{%
2883    \setbox0\hbox{d}\dimen@=\ht0
2884    \advance\dimen@1ex
2885    \dimen@.45\dimen@
2886    \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2887    \advance\dimen@ii.5ex
2888    \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2889 \def\DDJ@{%
2890    \setbox0\hbox{D}\dimen@=.55\ht0
2891    \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2892    \advance\dimen@ii.15ex %              correction for the dash position
2893    \advance\dimen@ii-.15\fontdimen7\font %     correction for cmtt font
2894    \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
2895    \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
2896 %
2897 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2898 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}
```

131

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

```
2899 \ProvideTextCommandDefault{\dj}{%
2900   \UseTextSymbol{OT1}{\dj}}
2901 \ProvideTextCommandDefault{\DJ}{%
2902   \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.

```
2903 \DeclareTextCommand{\SS}{OT1}{SS}
2904 \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
```
2905 \ProvideTextCommandDefault{\glq}{%
2906   \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}
```

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

```
2907 \ProvideTextCommand{\grq}{T1}{%
2908   \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}
2909 \ProvideTextCommand{\grq}{TU}{%
2910   \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
2911 \ProvideTextCommand{\grq}{OT1}{%
2912   \save@sf@q{\kern-.0125em
2913     \textormath{\textquoteleft}{\mbox{\textquoteleft}}%
2914     \kern.07em\relax}}
2915 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}\grq}
```

\glqq   The 'german' double quotes.
\grqq
```
2916 \ProvideTextCommandDefault{\glqq}{%
2917   \textormath{\quotedblbase}{\mbox{\quotedblbase}}}
```

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

```
2918 \ProvideTextCommand{\grqq}{T1}{%
2919   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2920 \ProvideTextCommand{\grqq}{TU}{%
2921   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2922 \ProvideTextCommand{\grqq}{OT1}{%
2923   \save@sf@q{\kern-.07em
2924     \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}%
2925     \kern.07em\relax}}
2926 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}
```

\flq   The 'french' single guillemets.
\frq
```
2927 \ProvideTextCommandDefault{\flq}{%
2928   \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
2929 \ProvideTextCommandDefault{\frq}{%
2930   \textormath{\guilsinglright}{\mbox{\guilsinglright}}}
```

\flqq   The 'french' double guillemets.
\frqq
```
2931 \ProvideTextCommandDefault{\flqq}{%
2932   \textormath{\guillemetleft}{\mbox{\guillemetleft}}}
2933 \ProvideTextCommandDefault{\frqq}{%
2934   \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`
`\umlautlow`
To be able to provide both positions of `\"` we provide two commands to switch the positioning, the default will be `\umlauthigh` (the normal positioning).

```
2935 \def\umlauthigh{%
2936   \def\bbl@umlauta##1{\leavevmode\bgroup%
2937      \expandafter\accent\csname\f@encoding dqpos\endcsname
2938      ##1\bbl@allowhyphens\egroup}%
2939   \let\bbl@umlaute\bbl@umlauta}
2940 \def\umlautlow{%
2941   \def\bbl@umlauta{\protect\lower@umlaut}}
2942 \def\umlautelow{%
2943   \def\bbl@umlaute{\protect\lower@umlaut}}
2944 \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.

```
2945 \expandafter\ifx\csname U@D\endcsname\relax
2946   \csname newdimen\endcsname\U@D
2947 \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.

```
2948 \def\lower@umlaut#1{%
2949   \leavevmode\bgroup
2950     \U@D 1ex%
2951     {\setbox\z@\hbox{%
2952       \expandafter\char\csname\f@encoding dqpos\endcsname}%
2953      \dimen@ -.45ex\advance\dimen@\ht\z@
2954      \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2955   \expandafter\accent\csname\f@encoding dqpos\endcsname
2956   \fontdimen5\font\U@D #1%
2957   \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).

```
2958 \AtBeginDocument{%
2959   \DeclareTextCompositeCommand{\"}{OT1}{a}{\bbl@umlauta{a}}%
2960   \DeclareTextCompositeCommand{\"}{OT1}{e}{\bbl@umlaute{e}}%
2961   \DeclareTextCompositeCommand{\"}{OT1}{i}{\bbl@umlaute{\i}}%
2962   \DeclareTextCompositeCommand{\"}{OT1}{\i}{\bbl@umlaute{\i}}%
2963   \DeclareTextCompositeCommand{\"}{OT1}{o}{\bbl@umlauta{o}}%
2964   \DeclareTextCompositeCommand{\"}{OT1}{u}{\bbl@umlauta{u}}%
2965   \DeclareTextCompositeCommand{\"}{OT1}{A}{\bbl@umlauta{A}}%
```

```
2966    \DeclareTextCompositeCommand{\"}{OT1}{E}{\bbl@umlaute{E}}%
2967    \DeclareTextCompositeCommand{\"}{OT1}{I}{\bbl@umlaute{I}}%
2968    \DeclareTextCompositeCommand{\"}{OT1}{O}{\bbl@umlauta{O}}%
2969    \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.

```
2970 \ifx\l@english\@undefined
2971    \chardef\l@english\z@
2972 \fi
2973 % The following is used to cancel rules in ini files (see Amharic).
2974 \ifx\l@unhyphenated\@undefined
2975    \newlanguage\l@unhyphenated
2976 \fi
```

9.13 Layout

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

```
2977 \bbl@trace{Bidi layout}
2978 \providecommand\IfBabelLayout[3]{#3}%
2979 \newcommand\BabelPatchSection[1]{%
2980    \@ifundefined{#1}{}{%
2981      \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
2982      \@namedef{#1}{%
2983        \@ifstar{\bbl@presec@s{#1}}%
2984                 {\@dblarg{\bbl@presec@x{#1}}}}}}
2985 \def\bbl@presec@x#1[#2]#3{%
2986    \bbl@exp{%
2987      \\\select@language@x{\bbl@main@language}%
2988      \\\bbl@cs{sspre@#1}%
2989      \\\bbl@cs{ss@#1}%
2990        [\\\foreignlanguage{\languagename}{\unexpanded{#2}}]%
2991        {\\\foreignlanguage{\languagename}{\unexpanded{#3}}}%
2992      \\\select@language@x{\languagename}}}
2993 \def\bbl@presec@s#1#2{%
2994    \bbl@exp{%
2995      \\\select@language@x{\bbl@main@language}%
2996      \\\bbl@cs{sspre@#1}%
2997      \\\bbl@cs{ss@#1}*%
2998        {\\\foreignlanguage{\languagename}{\unexpanded{#2}}}%
2999      \\\select@language@x{\languagename}}}
3000 \IfBabelLayout{sectioning}%
3001    {\BabelPatchSection{part}%
3002     \BabelPatchSection{chapter}%
3003     \BabelPatchSection{section}%
3004     \BabelPatchSection{subsection}%
3005     \BabelPatchSection{subsubsection}%
3006     \BabelPatchSection{paragraph}%
3007     \BabelPatchSection{subparagraph}%
3008     \def\babel@toc#1{%
3009       \select@language@x{\bbl@main@language}}}{}
3010 \IfBabelLayout{captions}%
3011    {\BabelPatchSection{caption}}{}
```

9.14 Load engine specific macros

```
3012 \bbl@trace{Input engine specific macros}
3013 \ifcase\bbl@engine
```

```
3014    \input txtbabel.def
3015 \or
3016    \input luababel.def
3017 \or
3018    \input xebabel.def
3019 \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.

```
3020 \bbl@trace{Creating languages and reading ini files}
3021 \newcommand\babelprovide[2][]{%
3022    \let\bbl@savelangname\languagename
3023    \edef\bbl@savelocaleid{\the\localeid}%
3024    % Set name and locale id
3025    \edef\languagename{#2}%
3026    % \global\@namedef{bbl@lcname@#2}{#2}%
3027    \bbl@id@assign
3028    \let\bbl@KVP@captions\@nil
3029    \let\bbl@KVP@date\@nil
3030    \let\bbl@KVP@import\@nil
3031    \let\bbl@KVP@main\@nil
3032    \let\bbl@KVP@script\@nil
3033    \let\bbl@KVP@language\@nil
3034    \let\bbl@KVP@hyphenrules\@nil
3035    \let\bbl@KVP@linebreaking\@nil
3036    \let\bbl@KVP@justification\@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}%
```

136

```
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
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
```

137

```
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
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  \ifx\bbl@KVP@justification\@nil\else
3219    \let\bbl@KVP@linebreaking\bbl@KVP@justification
3220  \fi
3221  \ifx\bbl@KVP@linebreaking\@nil\else
3222    \bbl@xin@{,\bbl@KVP@linebreaking,}{,elongated,kashida,cjk,unhyphenated,}%
3223    \ifin@
3224      \bbl@csarg\xdef
3225        {lnbrk@\languagename}{\expandafter\@car\bbl@KVP@linebreaking\@nil}%
3226    \fi
3227  \fi
3228  \bbl@xin@{/e}{/\bbl@cl{lnbrk}}%
3229  \ifin@\else\bbl@xin@{/k}{/\bbl@cl{lnbrk}}\fi
3230  \ifin@\bbl@arabicjust\fi
3231  % == Line breaking: hyphenate.other.locale/.script==
3232  \ifx\bbl@lbkflag\@empty
3233    \bbl@ifunset{bbl@hyotl@\languagename}{}%
3234      {\bbl@csarg\bbl@replace{hyotl@\languagename}{ }{,}%
3235       \bbl@startcommands*{\languagename}{}%
3236         \bbl@csarg\bbl@foreach{hyotl@\languagename}{%
3237           \ifcase\bbl@engine
3238             \ifnum##1<257
3239               \SetHyphenMap{\BabelLower{##1}{##1}}%
3240             \fi
3241           \else
3242             \SetHyphenMap{\BabelLower{##1}{##1}}%
3243           \fi}%
```

138

```
3244        \bbl@endcommands}%
3245     \bbl@ifunset{bbl@hyots@\languagename}{}%
3246       {\bbl@csarg\bbl@replace{hyots@\languagename}{ }{,}%
3247        \bbl@csarg\bbl@foreach{hyots@\languagename}{%
3248          \ifcase\bbl@engine
3249            \ifnum##1<257
3250              \global\lccode##1=##1\relax
3251            \fi
3252          \else
3253            \global\lccode##1=##1\relax
3254          \fi}}%
3255   \fi
3256   % == Counters: maparabic ==
3257   % Native digits, if provided in ini (TeX level, xe and lua)
3258   \ifcase\bbl@engine\else
3259     \bbl@ifunset{bbl@dgnat@\languagename}{}%
3260       {\expandafter\ifx\csname bbl@dgnat@\languagename\endcsname\@empty\else
3261        \expandafter\expandafter\expandafter
3262        \bbl@setdigits\csname bbl@dgnat@\languagename\endcsname
3263        \ifx\bbl@KVP@maparabic\@nil\else
3264          \ifx\bbl@latinarabic\@undefined
3265            \expandafter\let\expandafter\@arabic
3266              \csname bbl@counter@\languagename\endcsname
3267          \else     % ie, if layout=counters, which redefines \@arabic
3268            \expandafter\let\expandafter\bbl@latinarabic
3269              \csname bbl@counter@\languagename\endcsname
3270          \fi
3271        \fi
3272      \fi}%
3273   \fi
3274   % == Counters: mapdigits ==
3275   % Native digits (lua level).
3276   \ifodd\bbl@engine
3277     \ifx\bbl@KVP@mapdigits\@nil\else
3278       \bbl@ifunset{bbl@dgnat@\languagename}{}%
3279         {\RequirePackage{luatexbase}%
3280          \bbl@activate@preotf
3281          \directlua{
3282            Babel = Babel or {}  %%% -> presets in luababel
3283            Babel.digits_mapped = true
3284            Babel.digits = Babel.digits or {}
3285            Babel.digits[\the\localeid] =
3286              table.pack(string.utfvalue('\bbl@cl{dgnat}'))
3287            if not Babel.numbers then
3288              function Babel.numbers(head)
3289                local LOCALE = luatexbase.registernumber'bbl@attr@locale'
3290                local GLYPH = node.id'glyph'
3291                local inmath = false
3292                for item in node.traverse(head) do
3293                  if not inmath and item.id == GLYPH then
3294                    local temp = node.get_attribute(item, LOCALE)
3295                    if Babel.digits[temp] then
3296                      local chr = item.char
3297                      if chr > 47 and chr < 58 then
3298                        item.char = Babel.digits[temp][chr-47]
3299                      end
3300                    end
3301                  elseif item.id == node.id'math' then
3302                    inmath = (item.subtype == 0)
```

```
3303                     end
3304                   end
3305                   return head
3306                 end
3307               end
3308           }}%
3309       \fi
3310     \fi
3311     % == Counters: alph, Alph ==
3312     % What if extras<lang> contains a \babel@save\@alph? It won't be
3313     % restored correctly when exiting the language, so we ignore
3314     % this change with the \bbl@alph@saved trick.
3315     \ifx\bbl@KVP@alph\@nil\else
3316       \toks@\expandafter\expandafter\expandafter{%
3317         \csname extras\languagename\endcsname}%
3318       \bbl@exp{%
3319         \def\<extras\languagename>{%
3320           \let\\\bbl@alph@saved\\\@alph
3321           \the\toks@
3322           \let\\\@alph\\\bbl@alph@saved
3323           \\\babel@save\\\@alph
3324           \let\\\@alph\<bbl@cntr@\bbl@KVP@alph @\languagename>}}%
3325     \fi
3326     \ifx\bbl@KVP@Alph\@nil\else
3327       \toks@\expandafter\expandafter\expandafter{%
3328         \csname extras\languagename\endcsname}%
3329       \bbl@exp{%
3330         \def\<extras\languagename>{%
3331           \let\\\bbl@Alph@saved\\\@Alph
3332           \the\toks@
3333           \let\\\@Alph\\\bbl@Alph@saved
3334           \\\babel@save\\\@Alph
3335           \let\\\@Alph\<bbl@cntr@\bbl@KVP@Alph @\languagename>}}%
3336     \fi
3337     % == require.babel in ini ==
3338     % To load or reload the babel-*.tex, if require.babel in ini
3339     \ifx\bbl@beforestart\relax\else  % But not in doc aux or body
3340       \bbl@ifunset{bbl@rqtex@\languagename}{}%
3341         {\expandafter\ifx\csname bbl@rqtex@\languagename\endcsname\@empty\else
3342           \let\BabelBeforeIni\@gobbletwo
3343           \chardef\atcatcode=\catcode`\@
3344           \catcode`\@=11\relax
3345           \bbl@input@texini{\bbl@cs{rqtex@\languagename}}%
3346           \catcode`\@=\atcatcode
3347           \let\atcatcode\relax
3348         \fi}%
3349     \fi
3350     % == Release saved transforms ==
3351     \bbl@release@transforms\relax % \relax closes the last item.
3352     % == main ==
3353     \ifx\bbl@KVP@main\@nil  % Restore only if not 'main'
3354       \let\languagename\bbl@savelangname
3355       \chardef\localeid\bbl@savelocaleid\relax
3356     \fi}
```

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

```
3357 \def\bbl@provide@new#1{%
3358   \@namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
3359   \@namedef{extras#1}{}%
```

```
3360    \@namedef{noextras#1}{}%
3361    \bbl@startcommands*{#1}{captions}%
3362      \ifx\bbl@KVP@captions\@nil %      and also if import, implicit
3363        \def\bbl@tempb##1{%              elt for \bbl@captionslist
3364          \ifx##1\@empty\else
3365            \bbl@exp{%
3366              \\\SetString\\##1{%
3367                \\\bbl@nocaption{\bbl@stripslash##1}{#1\bbl@stripslash##1}}}%
3368            \expandafter\bbl@tempb
3369          \fi}%
3370        \expandafter\bbl@tempb\bbl@captionslist\@empty
3371      \else
3372        \ifx\bbl@initoload\relax
3373          \bbl@read@ini{\bbl@KVP@captions}2%  % Here letters cat = 11
3374        \else
3375          \bbl@read@ini{\bbl@initoload}2%     % Same
3376        \fi
3377      \fi
3378    \StartBabelCommands*{#1}{date}%
3379      \ifx\bbl@KVP@import\@nil
3380        \bbl@exp{%
3381          \\\SetString\\\today{\\\bbl@nocaption{today}{#1today}}}%
3382      \else
3383        \bbl@savetoday
3384        \bbl@savedate
3385      \fi
3386    \bbl@endcommands
3387    \bbl@load@basic{#1}%
3388    % == hyphenmins == (only if new)
3389    \bbl@exp{%
3390      \gdef\<#1hyphenmins>{%
3391        {\bbl@ifunset{bbl@lfthm@#1}{2}{\bbl@cs{lfthm@#1}}}%
3392        {\bbl@ifunset{bbl@rgthm@#1}{3}{\bbl@cs{rgthm@#1}}}}}%
3393    % == hyphenrules ==
3394    \bbl@provide@hyphens{#1}%
3395    % == frenchspacing == (only if new)
3396    \bbl@ifunset{bbl@frspc@#1}{}%
3397      {\edef\bbl@tempa{\bbl@cl{frspc}}%
3398       \edef\bbl@tempa{\expandafter\@car\bbl@tempa\@nil}%
3399       \if u\bbl@tempa          % do nothing
3400       \else\if n\bbl@tempa     % non french
3401         \expandafter\bbl@add\csname extras#1\endcsname{%
3402           \let\bbl@elt\bbl@fs@elt@i
3403           \bbl@fs@chars}%
3404       \else\if y\bbl@tempa     % french
3405         \expandafter\bbl@add\csname extras#1\endcsname{%
3406           \let\bbl@elt\bbl@fs@elt@ii
3407           \bbl@fs@chars}%
3408       \fi\fi\fi}%
3409    %
3410    \ifx\bbl@KVP@main\@nil\else
3411      \expandafter\main@language\expandafter{#1}%
3412    \fi}
3413 % A couple of macros used above, to avoid hashes #######...
3414 \def\bbl@fs@elt@i#1#2#3{%
3415    \ifnum\sfcode`#1=#2\relax
3416      \babel@savevariable{\sfcode`#1}%
3417      \sfcode`#1=#3\relax
3418    \fi}%
```

141

```
3419 \def\bbl@fs@elt@ii#1#2#3{%
3420   \ifnum\sfcode`#1=#3\relax
3421     \babel@savevariable{\sfcode`#1}%
3422     \sfcode`#1=#2\relax
3423   \fi}%
3424 %
3425 \def\bbl@provide@renew#1{%
3426   \ifx\bbl@KVP@captions\@nil\else
3427     \StartBabelCommands*{#1}{captions}%
3428       \bbl@read@ini{\bbl@KVP@captions}2%   % Here all letters cat = 11
3429     \EndBabelCommands
3430   \fi
3431   \ifx\bbl@KVP@import\@nil\else
3432     \StartBabelCommands*{#1}{date}%
3433       \bbl@savetoday
3434       \bbl@savedate
3435     \EndBabelCommands
3436   \fi
3437   % == hyphenrules ==
3438   \ifx\bbl@lbkflag\@empty
3439     \bbl@provide@hyphens{#1}%
3440   \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.)

```
3441 \def\bbl@load@basic#1{%
3442   \bbl@ifunset{bbl@inidata@\languagename}{}%
3443     {\getlocaleproperty\bbl@tempa{\languagename}{identification/load.level}%
3444      \ifcase\bbl@tempa
3445        \bbl@csarg\let{lname@\languagename}\relax
3446      \fi}%
3447   \bbl@ifunset{bbl@lname@#1}%
3448     {\def\BabelBeforeIni##1##2{%
3449        \begingroup
3450          \let\bbl@ini@captions@aux\@gobbletwo
3451          \def\bbl@inidate ####1.####2.####3.####4\relax ####5####6{}%
3452          \bbl@read@ini{##1}1%
3453          \ifx\bbl@initoload\relax\endinput\fi
3454        \endgroup}%
3455      \begingroup      % boxed, to avoid extra spaces:
3456        \ifx\bbl@initoload\relax
3457          \bbl@input@texini{#1}%
3458        \else
3459          \setbox\z@\hbox{\BabelBeforeIni{\bbl@initoload}{}}%
3460        \fi
3461      \endgroup}%
3462     {}}
```

The hyphenrules option is handled with an auxiliary macro.

```
3463 \def\bbl@provide@hyphens#1{%
3464   \let\bbl@tempa\relax
3465   \ifx\bbl@KVP@hyphenrules\@nil\else
3466     \bbl@replace\bbl@KVP@hyphenrules{ }{,}%
3467     \bbl@foreach\bbl@KVP@hyphenrules{%
3468       \ifx\bbl@tempa\relax     %  if not yet found
3469         \bbl@ifsamestring{##1}{+}%
3470           {{\bbl@exp{\\\addlanguage\<l@##1>}}}%
3471           {}%
```

142

```
3472        \bbl@ifunset{l@##1}%
3473          {}%
3474          {\bbl@exp{\let\bbl@tempa\<l@##1>}}%
3475      \fi}%
3476  \fi
3477  \ifx\bbl@tempa\relax %          if no opt or no language in opt found
3478    \ifx\bbl@KVP@import\@nil
3479      \ifx\bbl@initoload\relax\else
3480        \bbl@exp{%              and hyphenrules is not empty
3481          \\\bbl@ifblank{\bbl@cs{hyphr@#1}}%
3482            {}%
3483            {\let\\\bbl@tempa\<l@\bbl@cl{hyphr}>}}%
3484      \fi
3485    \else % if importing
3486      \bbl@exp{%                  and hyphenrules is not empty
3487        \\\bbl@ifblank{\bbl@cs{hyphr@#1}}%
3488          {}%
3489          {\let\\\bbl@tempa\<l@\bbl@cl{hyphr}>}}%
3490    \fi
3491  \fi
3492  \bbl@ifunset{bbl@tempa}%        ie, relax or undefined
3493    {\bbl@ifunset{l@#1}%          no hyphenrules found - fallback
3494      {\bbl@exp{\\\adddialect\<l@#1>\language}}%
3495      {}}%                        so, l@<lang> is ok - nothing to do
3496    {\bbl@exp{\\\adddialect\<l@#1>\bbl@tempa}}}% found in opt list or ini
```

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

```
3497 \def\bbl@input@texini#1{%
3498   \bbl@bsphack
3499   \bbl@exp{%
3500     \catcode`\\\%=14 \catcode`\\\\=0
3501     \catcode`\\\{=1  \catcode`\\\}=2
3502     \lowercase{\\\InputIfFileExists{babel-#1.tex}{}{}}%
3503     \catcode`\\\%=\the\catcode`\%\relax
3504     \catcode`\\\\=\the\catcode`\\\relax
3505     \catcode`\\\{=\the\catcode`\{\relax
3506     \catcode`\\\}=\the\catcode`\}\relax}%
3507   \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.

```
3508 \def\bbl@iniline#1\bbl@iniline{%
3509   \@ifnextchar[\bbl@inisect{\@ifnextchar;\bbl@iniskip\bbl@inistore}#1\@@}% ]
3510 \def\bbl@inisect[#1]#2\@@{\def\bbl@section{#1}}%
3511 \def\bbl@iniskip#1\@@{}%      if starts with ;
3512 \def\bbl@inistore#1=#2\@@{%      full (default)
3513   \bbl@trim@def\bbl@tempa{#1}%
3514   \bbl@trim\toks@{#2}%
3515   \bbl@ifunset{bbl@KVP@\bbl@section/\bbl@tempa}%
3516     {\bbl@exp{%
3517       \\\g@addto@macro\\\bbl@inidata{%
3518         \\\bbl@elt{\bbl@section}{\bbl@tempa}{\the\toks@}}}}%
3519     {}}%
3520 \def\bbl@inistore@min#1=#2\@@{%  minimal (maybe set in \bbl@read@ini)
3521   \bbl@trim@def\bbl@tempa{#1}%
3522   \bbl@trim\toks@{#2}%
3523   \bbl@xin@{.identification.}{.\bbl@section.}%
3524   \ifin@
```

```
3525     \bbl@exp{\\\g@addto@macro\\\bbl@inidata{%
3526       \\\bbl@elt{identification}{\bbl@tempa}{\the\toks@}}}%
3527   \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.

```
3528 \ifx\bbl@readstream\@undefined
3529   \csname newread\endcsname\bbl@readstream
3530 \fi
3531 \def\bbl@read@ini#1#2{%
3532   \openin\bbl@readstream=babel-#1.ini
3533   \ifeof\bbl@readstream
3534     \bbl@error
3535       {There is no ini file for the requested language\\%
3536        (#1). Perhaps you misspelled it or your installation\\%
3537        is not complete.}%
3538       {Fix the name or reinstall babel.}%
3539   \else
3540     % Store ini data in \bbl@inidata
3541     \catcode`\[=12 \catcode`\]=12 \catcode`\==12 \catcode`\&=12
3542     \catcode`\;=12 \catcode`\|=12 \catcode`\%=14 \catcode`\-=12
3543     \bbl@info{Importing
3544                 \ifcase#2font and identification \or basic \fi
3545                  data for \languagename\\%
3546             from babel-#1.ini. Reported}%
3547     \ifnum#2=\z@
3548       \global\let\bbl@inidata\@empty
3549       \let\bbl@inistore\bbl@inistore@min     % Remember it's local
3550     \fi
3551     \def\bbl@section{identification}%
3552     \bbl@exp{\\\bbl@inistore tag.ini=#1\\\@@}%
3553     \bbl@inistore load.level=#2\@@
3554     \loop
3555     \if T\ifeof\bbl@readstream F\fi T\relax % Trick, because inside \loop
3556       \endlinechar\m@ne
3557       \read\bbl@readstream to \bbl@line
3558       \endlinechar`\^^M
3559       \ifx\bbl@line\@empty\else
3560         \expandafter\bbl@iniline\bbl@line\bbl@iniline
3561       \fi
3562     \repeat
3563     % Process stored data
3564     \bbl@csarg\xdef{lini@\languagename}{#1}%
3565     \let\bbl@savestrings\@empty
3566     \let\bbl@savetoday\@empty
3567     \let\bbl@savedate\@empty
3568     \def\bbl@elt##1##2##3{%
3569       \def\bbl@section{##1}%
3570       \in@{=date.}{=##1}% Find a better place
3571       \ifin@
3572         \bbl@ini@calendar{##1}%
3573       \fi
3574       \global\bbl@csarg\let{bbl@KVP@##1/##2}\relax
3575       \bbl@ifunset{bbl@inikv@##1}{}%
3576         {\csname bbl@inikv@##1\endcsname{##2}{##3}}}%
```

```
3577        \bbl@inidata
3578        % 'Export' data
3579        \bbl@ini@exports{#2}%
3580        \global\bbl@csarg\let{inidata@\languagename}\bbl@inidata
3581        \global\let\bbl@inidata\@empty
3582        \bbl@exp{\\\bbl@add@list\\\bbl@ini@loaded{\languagename}}%
3583        \bbl@toglobal\bbl@ini@loaded
3584    \fi}
```

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

```
3585 \def\bbl@ini@calendar#1{%
3586 \lowercase{\def\bbl@tempa{=#1=}}%
3587 \bbl@replace\bbl@tempa{=date.gregorian}{}%
3588 \bbl@replace\bbl@tempa{=date.}{}%
3589 \in@{.licr=}{#1=}%
3590 \ifin@
3591    \ifcase\bbl@engine
3592        \bbl@replace\bbl@tempa{.licr=}{}%
3593    \else
3594        \let\bbl@tempa\relax
3595    \fi
3596 \fi
3597 \ifx\bbl@tempa\relax\else
3598    \bbl@replace\bbl@tempa{=}{}%
3599    \bbl@exp{%
3600        \def\<bbl@inikv@#1>####1####2{%
3601            \\\bbl@inidate####1...\relax{####2}{\bbl@tempa}}}%
3602 \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).

```
3603 \def\bbl@renewinikey#1/#2\@@#3{%
3604    \edef\bbl@tempa{\zap@space #1 \@empty}%    section
3605    \edef\bbl@tempb{\zap@space #2 \@empty}%    key
3606    \bbl@trim\toks@{#3}%                       value
3607    \bbl@exp{%
3608        \global\let\<bbl@KVP@\bbl@tempa/\bbl@tempb>\\\@empty % just a flag
3609        \\\g@addto@macro\\\bbl@inidata{%
3610            \\\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.

```
3611 \def\bbl@exportkey#1#2#3{%
3612    \bbl@ifunset{bbl@@kv@#2}%
3613        {\bbl@csarg\gdef{#1@\languagename}{#3}}%
3614        {\expandafter\ifx\csname bbl@@kv@#2\endcsname\@empty
3615            \bbl@csarg\gdef{#1@\languagename}{#3}%
3616        \else
3617            \bbl@exp{\global\let\<bbl@#1@\languagename>\<bbl@@kv@#2>}%
3618        \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.

```
3619 \def\bbl@iniwarning#1{%
3620    \bbl@ifunset{bbl@@kv@identification.warning#1}{}%
3621        {\bbl@warning{%
```

145

```
3622        From babel-\bbl@cs{lini@\languagename}.ini:\\%
3623        \bbl@cs{@kv@identification.warning#1}\\%
3624        Reported }}}
3625 %
3626 \let\bbl@release@transforms\@empty
3627 %
3628 \def\bbl@ini@exports#1{%
3629   % Identification always exported
3630   \bbl@iniwarning{}%
3631   \ifcase\bbl@engine
3632     \bbl@iniwarning{.pdflatex}%
3633   \or
3634     \bbl@iniwarning{.lualatex}%
3635   \or
3636     \bbl@iniwarning{.xelatex}%
3637   \fi%
3638   \bbl@exportkey{elname}{identification.name.english}{}%
3639   \bbl@exp{\\\bbl@exportkey{lname}{identification.name.opentype}%
3640     {\csname bbl@elname@\languagename\endcsname}}%
3641   \bbl@exportkey{tbcp}{identification.tag.bcp47}{}%
3642   \bbl@exportkey{lbcp}{identification.language.tag.bcp47}{}%
3643   \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
3644   \bbl@exportkey{esname}{identification.script.name}{}%
3645   \bbl@exp{\\\bbl@exportkey{sname}{identification.script.name.opentype}%
3646     {\csname bbl@esname@\languagename\endcsname}}%
3647   \bbl@exportkey{sbcp}{identification.script.tag.bcp47}{}%
3648   \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
3649   % Also maps bcp47 -> languagename
3650   \ifbbl@bcptoname
3651     \bbl@csarg\xdef{bcp@map@\bbl@cl{tbcp}}{\languagename}%
3652   \fi
3653   % Conditional
3654   \ifnum#1>\z@          % 0 = only info, 1, 2 = basic, (re)new
3655     \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
3656     \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
3657     \bbl@exportkey{lfthm}{typography.lefthyphenmin}{2}%
3658     \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
3659     \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
3660     \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
3661     \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
3662     \bbl@exportkey{intsp}{typography.intraspace}{}%
3663     \bbl@exportkey{chrng}{characters.ranges}{}%
3664     \bbl@exportkey{dgnat}{numbers.digits.native}{}%
3665     \bbl@exportkey{frspc}{typography.frenchspacing}{u}%
3666     \ifnum#1=\tw@          % only (re)new
3667       \bbl@exportkey{rqtex}{identification.require.babel}{}%
3668       \bbl@toglobal\bbl@savetoday
3669       \bbl@toglobal\bbl@savedate
3670       \bbl@savestrings
3671     \fi
3672   \fi}
```

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

```
3673 \def\bbl@inikv#1#2{%       key=value
3674   \toks@{#2}%               This hides #'s from ini values
3675   \bbl@csarg\edef{@kv@\bbl@section.#1}{\the\toks@}}
```

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

```
3676 \let\bbl@inikv@identification\bbl@inikv
```

146

```
3677 \let\bbl@inikv@typography\bbl@inikv
3678 \let\bbl@inikv@characters\bbl@inikv
3679 \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'.

```
3680 \def\bbl@inikv@counters#1#2{%
3681   \bbl@ifsamestring{#1}{digits}%
3682     {\bbl@error{The counter name 'digits' is reserved for mapping\\%
3683                 decimal digits}%
3684                {Use another name.}}%
3685     {}%
3686   \def\bbl@tempc{#1}%
3687   \bbl@trim@def{\bbl@tempb*}{#2}%
3688   \in@{.1$}{#1$}%
3689   \ifin@
3690     \bbl@replace\bbl@tempc{.1}{}%
3691     \bbl@csarg\protected@xdef{cntr@\bbl@tempc @\languagename}{%
3692       \noexpand\bbl@alphnumeral{\bbl@tempc}}%
3693   \fi
3694   \in@{.F.}{#1}%
3695   \ifin@\else\in@{.S.}{#1}\fi
3696   \ifin@
3697     \bbl@csarg\protected@xdef{cntr@#1@\languagename}{\bbl@tempb*}%
3698   \else
3699     \toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa
3700     \expandafter\bbl@buildifcase\bbl@tempb* \\ % Space after \\
3701     \bbl@csarg{\global\expandafter\let}{cntr@#1@\languagename}\bbl@tempa
3702   \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.

```
3703 \ifcase\bbl@engine
3704   \bbl@csarg\def{inikv@captions.licr}#1#2{%
3705     \bbl@ini@captions@aux{#1}{#2}}
3706 \else
3707   \def\bbl@inikv@captions#1#2{%
3708     \bbl@ini@captions@aux{#1}{#2}}
3709 \fi
```

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

```
3710 \def\bbl@ini@captions@template#1#2{% string language tempa=capt-name
3711   \bbl@replace\bbl@tempa{.template}{}%
3712   \def\bbl@toreplace{#1{}}%
3713   \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
3714   \bbl@replace\bbl@toreplace{[[}{\csname}%
3715   \bbl@replace\bbl@toreplace{[}{\csname the}%
3716   \bbl@replace\bbl@toreplace{]]}{name\endcsname{}}%
3717   \bbl@replace\bbl@toreplace{]}{\endcsname{}}%
3718   \bbl@xin@{,\bbl@tempa,}{,chapter,appendix,part,}%
3719   \ifin@
3720     \@nameuse{bbl@patch\bbl@tempa}%
3721     \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
3722   \fi
3723   \bbl@xin@{,\bbl@tempa,}{,figure,table,}%
3724   \ifin@
3725     \toks@\expandafter{\bbl@toreplace}%
```

147

```
3726      \bbl@exp{\gdef\<fnum@\bbl@tempa>{\the\toks@}}%
3727    \fi}
3728 \def\bbl@ini@captions@aux#1#2{%
3729    \bbl@trim@def\bbl@tempa{#1}%
3730    \bbl@xin@{.template}{\bbl@tempa}%
3731    \ifin@
3732      \bbl@ini@captions@template{#2}\languagename
3733    \else
3734      \bbl@ifblank{#2}%
3735        {\bbl@exp{%
3736          \toks@{\\\bbl@nocaption{\bbl@tempa}{\languagename\bbl@tempa name}}}}%
3737        {\bbl@trim\toks@{#2}}%
3738      \bbl@exp{%
3739        \\\bbl@add\\\bbl@savestrings{%
3740          \\\SetString\<\bbl@tempa name>{\the\toks@}}}%
3741      \toks@\expandafter{\bbl@captionslist}%
3742      \bbl@exp{\\\in@{\<\bbl@tempa name>}{\the\toks@}}%
3743      \ifin@\else
3744        \bbl@exp{%
3745          \\\bbl@add\<bbl@extracaps@\languagename>{\<\bbl@tempa name>}%
3746          \\\bbl@toglobal\<bbl@extracaps@\languagename>}%
3747      \fi
3748    \fi}
```

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

```
3749 \def\bbl@list@the{%
3750    part,chapter,section,subsection,subsubsection,paragraph,%
3751    subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,%
3752    table,page,footnote,mpfootnote,mpfn}
3753 \def\bbl@map@cnt#1{%  #1:roman,etc, // #2:enumi,etc
3754    \bbl@ifunset{bbl@map@#1@\languagename}%
3755      {\@nameuse{#1}}%
3756      {\@nameuse{bbl@map@#1@\languagename}}}
3757 \def\bbl@inikv@labels#1#2{%
3758    \in@{.map}{#1}%
3759    \ifin@
3760      \ifx\bbl@KVP@labels\@nil\else
3761        \bbl@xin@{ map }{ \bbl@KVP@labels\space}%
3762        \ifin@
3763          \def\bbl@tempc{#1}%
3764          \bbl@replace\bbl@tempc{.map}{}%
3765          \in@{,#2,}{,arabic,roman,Roman,alph,Alph,fnsymbol,}%
3766          \bbl@exp{%
3767            \gdef\<bbl@map@\bbl@tempc @\languagename>%
3768              {\ifin@\<#2>\else\\\localecounter{#2}\fi}}%
3769          \bbl@foreach\bbl@list@the{%
3770            \bbl@ifunset{the##1}{}%
3771              {\bbl@exp{\let\\\bbl@tempd\<the##1>}%
3772               \bbl@exp{%
3773                 \\\bbl@sreplace\<the##1>%
3774                   {\<\bbl@tempc>{##1}}{\\\bbl@map@cnt{\bbl@tempc}{##1}}%
3775                 \\\bbl@sreplace\<the##1>%
3776                   {\<\@empty @\bbl@tempc>\<c@##1>}{\\\bbl@map@cnt{\bbl@tempc}{##1}}}%
3777              \expandafter\ifx\csname the##1\endcsname\bbl@tempd\else
3778                \toks@\expandafter\expandafter\expandafter{%
3779                  \csname the##1\endcsname}%
3780                \expandafter\xdef\csname the##1\endcsname{{\the\toks@}}%
3781              \fi}}%
3782        \fi
```

```
3783      \fi
3784  %
3785  \else
3786    %
3787    % The following code is still under study. You can test it and make
3788    % suggestions. Eg, enumerate.2 = ([enumi]).([enumii]). It's
3789    % language dependent.
3790    \in@{enumerate.}{#1}%
3791    \ifin@
3792      \def\bbl@tempa{#1}%
3793      \bbl@replace\bbl@tempa{enumerate.}{}%
3794      \def\bbl@toreplace{#2}%
3795      \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
3796      \bbl@replace\bbl@toreplace{[}{\csname the}%
3797      \bbl@replace\bbl@toreplace{]}{\endcsname{}}%
3798      \toks@\expandafter{\bbl@toreplace}%
3799      \bbl@exp{%
3800        \\\bbl@add\<extras\languagename>{%
3801          \\\babel@save\<labelenum\romannumeral\bbl@tempa>%
3802          \def\<labelenum\romannumeral\bbl@tempa>{\the\toks@}}%
3803        \\\bbl@toglobal\<extras\languagename>}%
3804    \fi
3805  \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.

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

```
3837    \let\bbl@patchpart\relax
3838 \else
3839    \def\bbl@patchpart{%
3840      \global\let\bbl@patchpart\relax
3841      \bbl@sreplace\@part
3842        {\partname\nobreakspace\thepart}%
3843        {\bbl@partformat}%
3844      \bbl@toglobal\@part
3845      \gdef\bbl@partformat{%
3846        \bbl@ifunset{bbl@partfmt@\languagename}%
3847          {\partname\nobreakspace\thepart}
3848          {\@nameuse{bbl@partfmt@\languagename}}}}
3849 \fi
```

Date. TODO. Document

```
3850 % Arguments are _not_ protected.
3851 \let\bbl@calendar\@empty
3852 \DeclareRobustCommand\localedate[1][]{\bbl@localedate{#1}}
3853 \def\bbl@localedate#1#2#3#4{%
3854    \begingroup
3855      \ifx\@empty#1\@empty\else
3856        \let\bbl@ld@calendar\@empty
3857        \let\bbl@ld@variant\@empty
3858        \edef\bbl@tempa{\zap@space#1 \@empty}%
3859        \def\bbl@tempb##1=##2\@@{\@namedef{bbl@ld@##1}{##2}}%
3860        \bbl@foreach\bbl@tempa{\bbl@tempb##1\@@}%
3861        \edef\bbl@calendar{%
3862          \bbl@ld@calendar
3863          \ifx\bbl@ld@variant\@empty\else
3864            .\bbl@ld@variant
3865          \fi}%
3866        \bbl@replace\bbl@calendar{gregorian}{}%
3867      \fi
3868      \bbl@cased
3869        {\@nameuse{bbl@date@\languagename @\bbl@calendar}{#2}{#3}{#4}}%
3870    \endgroup}
3871 % eg: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
3872 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
3873    \bbl@trim@def\bbl@tempa{#1.#2}%
3874    \bbl@ifsamestring{\bbl@tempa}{months.wide}%      to savedate
3875      {\bbl@trim@def\bbl@tempa{#3}%
3876       \bbl@trim\toks@{#5}%
3877       \@temptokena\expandafter{\bbl@savedate}%
3878       \bbl@exp{%   Reverse order - in ini last wins
3879         \def\\\bbl@savedate{%
3880           \\\SetString\<month\romannumeral\bbl@tempa#6name>{\the\toks@}%
3881           \the\@temptokena}}%
3882      {\bbl@ifsamestring{\bbl@tempa}{date.long}%      defined now
3883        {\lowercase{\def\bbl@tempb{#6}}%
3884         \bbl@trim@def\bbl@toreplace{#5}%
3885         \bbl@TG@@date
3886         \bbl@ifunset{bbl@date@\languagename @}%
3887           {\global\bbl@csarg\let{date@\languagename @}\bbl@toreplace
3888           % TODO. Move to a better place.
3889            \bbl@exp{%
3890              \gdef\<\languagename date>{\\\protect\<\languagename date >}%
3891              \gdef\<\languagename date >####1####2####3{%
3892                \\\bbl@usedategrouptrue
3893                \<bbl@ensure@\languagename>{%
```

150

```
3894                 \\\localedate{####1}{####2}{####3}}}%
3895            \\\bbl@add\\\bbl@savetoday{%
3896              \\\SetString\\\today{%
3897                \<\languagename date>%
3898                  {\\\the\year}{\\\the\month}{\\\the\day}}}}}%
3899          {}%
3900        \ifx\bbl@tempb\@empty\else
3901          \global\bbl@csarg\let{date@\languagename @\bbl@tempb}\bbl@toreplace
3902        \fi}%
3903      {}}}
```

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.

```
3904 \let\bbl@calendar\@empty
3905 \newcommand\BabelDateSpace{\nobreakspace}
3906 \newcommand\BabelDateDot{.\@}  % TODO. \let instead of repeating
3907 \newcommand\BabelDated[1]{{\number#1}}
3908 \newcommand\BabelDatedd[1]{{\ifnum#1<10 0\fi\number#1}}
3909 \newcommand\BabelDateM[1]{{\number#1}}
3910 \newcommand\BabelDateMM[1]{{\ifnum#1<10 0\fi\number#1}}
3911 \newcommand\BabelDateMMMM[1]{{%
3912   \csname month\romannumeral#1\bbl@calendar name\endcsname}}%
3913 \newcommand\BabelDatey[1]{{\number#1}}%
3914 \newcommand\BabelDateyy[1]{{%
3915   \ifnum#1<10 0\number#1 %
3916   \else\ifnum#1<100 \number#1 %
3917   \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3918   \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
3919   \else
3920     \bbl@error
3921       {Currently two-digit years are restricted to the\\
3922        range 0-9999.}%
3923       {There is little you can do. Sorry.}%
3924   \fi\fi\fi\fi}}
3925 \newcommand\BabelDateyyyy[1]{{\number#1}} % FIXME - add leading 0
3926 \def\bbl@replace@finish@iii#1{%
3927   \bbl@exp{\def\\#1####1####2####3{\the\toks@}}}
3928 \def\bbl@TG@@date{%
3929   \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace{}}%
3930   \bbl@replace\bbl@toreplace{[.]}{\BabelDateDot{}}%
3931   \bbl@replace\bbl@toreplace{[d]}{\BabelDated{####3}}%
3932   \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{####3}}%
3933   \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{####2}}%
3934   \bbl@replace\bbl@toreplace{[MM]}{\BabelDateMM{####2}}%
3935   \bbl@replace\bbl@toreplace{[MMMM]}{\BabelDateMMMM{####2}}%
3936   \bbl@replace\bbl@toreplace{[y]}{\BabelDatey{####1}}%
3937   \bbl@replace\bbl@toreplace{[yy]}{\BabelDateyy{####1}}%
3938   \bbl@replace\bbl@toreplace{[yyyy]}{\BabelDateyyyy{####1}}%
3939   \bbl@replace\bbl@toreplace{[y|}{\bbl@datecntr[####1|}%
3940   \bbl@replace\bbl@toreplace{[m|}{\bbl@datecntr[####2|}%
3941   \bbl@replace\bbl@toreplace{[d|}{\bbl@datecntr[####3|}%
3942 % Note after \bbl@replace \toks@ contains the resulting string.
3943 % TODO - Using this implicit behavior doesn't seem a good idea.
3944   \bbl@replace@finish@iii\bbl@toreplace}
3945 \def\bbl@datecntr{\expandafter\bbl@xdatecntr\expandafter}
3946 \def\bbl@xdatecntr[#1|#2]{\localenumeral{#2}{#1}}
```

Transforms.

```
3947 \let\bbl@release@transforms\@empty
3948 \@namedef{bbl@inikv@transforms.prehyphenation}{%
3949   \bbl@transforms\babelprehyphenation}
3950 \@namedef{bbl@inikv@transforms.posthyphenation}{%
3951   \bbl@transforms\babelposthyphenation}
3952 \def\bbl@transforms@aux#1#2#3,#4\relax{#1{#2}{#3}{#4}}
3953 \begingroup
3954   \catcode`\%=12
3955   \catcode`\&=14
3956   \gdef\bbl@transforms#1#2#3{&%
3957     \ifx\bbl@KVP@transforms\@nil\else
3958       \directlua{
3959         str = [==[#2]==]
3960         str = str:gsub('%.%d+%.%d+$', '')
3961         tex.print([[\def\string\babeltempa{]] .. str .. [[}]])
3962       }&%
3963       \bbl@xin@{,\babeltempa,}{,\bbl@KVP@transforms,}&%
3964       \ifin@
3965         \in@{.0$}{#2$}&%
3966         \ifin@
3967           \g@addto@macro\bbl@release@transforms{&%
3968             \relax\bbl@transforms@aux#1{\languagename}{#3}}&%
3969         \else
3970           \g@addto@macro\bbl@release@transforms{, {#3}}&%
3971         \fi
3972       \fi
3973     \fi}
3974 \endgroup
```

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

```
3975 \def\bbl@provide@lsys#1{%
3976   \bbl@ifunset{bbl@lname@#1}%
3977     {\bbl@load@info{#1}}%
3978     {}%
3979   \bbl@csarg\let{lsys@#1}\@empty
3980   \bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}{}%
3981   \bbl@ifunset{bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{}%
3982   \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
3983   \bbl@ifunset{bbl@lname@#1}{}%
3984     {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
3985   \ifcase\bbl@engine\or\or
3986     \bbl@ifunset{bbl@prehc@#1}{}%
3987       {\bbl@exp{\\\bbl@ifblank{\bbl@cs{prehc@#1}}}%
3988         {}%
3989         {\ifx\bbl@xenohyph\@undefined
3990           \let\bbl@xenohyph\bbl@xenohyph@d
3991           \ifx\AtBeginDocument\@notprerr
3992             \expandafter\@secondoftwo  % to execute right now
3993           \fi
3994           \AtBeginDocument{%
3995             \expandafter\bbl@add
3996             \csname selectfont \endcsname{\bbl@xenohyph}%
3997             \expandafter\selectlanguage\expandafter{\languagename}%
3998             \expandafter\bbl@toglobal\csname selectfont \endcsname}%
3999         \fi}}%
4000   \fi
4001   \bbl@csarg\bbl@toglobal{lsys@#1}}
4002 \def\bbl@xenohyph@d{%
```

```
4003    \bbl@ifset{bbl@prehc@\languagename}%
4004      {\ifnum\hyphenchar\font=\defaulthyphenchar
4005         \iffontchar\font\bbl@cl{prehc}\relax
4006           \hyphenchar\font\bbl@cl{prehc}\relax
4007         \else\iffontchar\font"200B
4008           \hyphenchar\font"200B
4009         \else
4010           \bbl@warning
4011             {Neither 0 nor ZERO WIDTH SPACE are available\\%
4012              in the current font, and therefore the hyphen\\%
4013              will be printed. Try changing the fontspec's\\%
4014              'HyphenChar' to another value, but be aware\\%
4015              this setting is not safe (see the manual)}%
4016           \hyphenchar\font\defaulthyphenchar
4017         \fi\fi
4018      \fi}%
4019      {\hyphenchar\font\defaulthyphenchar}}
4020  % \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).

```
4021 \def\bbl@load@info#1{%
4022    \def\BabelBeforeIni##1##2{%
4023      \begingroup
4024      \bbl@read@ini{##1}0%
4025      \endinput         % babel- .tex may contain onlypreamble's
4026      \endgroup}%              boxed, to avoid extra spaces:
4027    {\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.

```
4028 \def\bbl@setdigits#1#2#3#4#5{%
4029    \bbl@exp{%
4030      \def\<\languagename digits>####1{%        ie, \langdigits
4031        \<bbl@digits@\languagename>####1\\\@nil}%
4032      \let\<bbl@cntr@digits@\languagename>\<\languagename digits>%
4033      \def\<\languagename counter>####1{%       ie, \langcounter
4034        \\\expandafter\<bbl@counter@\languagename>%
4035        \\\csname c@####1\endcsname}%
4036      \def\<bbl@counter@\languagename>####1{% ie, \bbl@counter@lang
4037        \\\expandafter\<bbl@digits@\languagename>%
4038        \\\number####1\\\@nil}}%
4039    \def\bbl@tempa##1##2##3##4##5{%
4040      \bbl@exp{%    Wow, quite a lot of hashes! :-(
4041        \def\<bbl@digits@\languagename>########1{%
4042          \\\ifx########1\\\@nil              % ie, \bbl@digits@lang
4043          \\\else
4044            \\\ifx0########1#1%
4045            \\\else\\\ifx1########1#2%
4046            \\\else\\\ifx2########1#3%
4047            \\\else\\\ifx3########1#4%
4048            \\\else\\\ifx4########1#5%
4049            \\\else\\\ifx5########1##1%
4050            \\\else\\\ifx6########1##2%
4051            \\\else\\\ifx7########1##3%
4052            \\\else\\\ifx8########1##4%
```

```
4053        \\\else\\\ifx9########1##5%
4054        \\\else########1%
4055        \\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi
4056        \\\expandafter\<bbl@digits@\languagename>%
4057      \\\fi}}}%
4058   \bbl@tempa}
```

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

```
4059 \def\bbl@buildifcase#1 {% Returns \bbl@tempa, requires \toks@={}
4060   \ifx\\#1%                % \\ before, in case #1 is multiletter
4061     \bbl@exp{%
4062       \def\\bbl@tempa####1{%
4063         \<ifcase>####1\space\the\toks@\<else>\\\@ctrerr\<fi>}}%
4064   \else
4065     \toks@\expandafter{\the\toks@\or #1}%
4066     \expandafter\bbl@buildifcase
4067   \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).

```
4068 \newcommand\localenumeral[2]{\bbl@cs{cntr@#1@\languagename}{#2}}
4069 \def\bbl@localecntr#1#2{\localenumeral{#2}{#1}}
4070 \newcommand\localecounter[2]{%
4071   \expandafter\bbl@localecntr
4072   \expandafter{\number\csname c@#2\endcsname}{#1}}
4073 \def\bbl@alphnumeral#1#2{%
4074   \expandafter\bbl@alphnumeral@i\number#2 76543210\@@{#1}}
4075 \def\bbl@alphnumeral@i#1#2#3#4#5#6#7#8\@@#9{%
4076   \ifcase\@car#8\@nil\or    % Currenty <10000, but prepared for bigger
4077     \bbl@alphnumeral@ii{#9}000000#1\or
4078     \bbl@alphnumeral@ii{#9}00000#1#2\or
4079     \bbl@alphnumeral@ii{#9}0000#1#2#3\or
4080     \bbl@alphnumeral@ii{#9}000#1#2#3#4\else
4081     \bbl@alphnum@invalid{>9999}%
4082   \fi}
4083 \def\bbl@alphnumeral@ii#1#2#3#4#5#6#7#8{%
4084   \bbl@ifunset{bbl@cntr@#1.F.\number#5#6#7#8@\languagename}%
4085     {\bbl@cs{cntr@#1.4@\languagename}#5%
4086      \bbl@cs{cntr@#1.3@\languagename}#6%
4087      \bbl@cs{cntr@#1.2@\languagename}#7%
4088      \bbl@cs{cntr@#1.1@\languagename}#8%
4089      \ifnum#6#7#8>\z@ % TODO. An ad hoc rule for Greek. Ugly.
4090        \bbl@ifunset{bbl@cntr@#1.S.321@\languagename}{}%
4091          {\bbl@cs{cntr@#1.S.321@\languagename}}%
4092      \fi}%
4093     {\bbl@cs{cntr@#1.F.\number#5#6#7#8@\languagename}}}
4094 \def\bbl@alphnum@invalid#1{%
4095   \bbl@error{Alphabetic numeral too large (#1)}%
4096     {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.

```
4097 \newcommand\localeinfo[1]{%
4098   \bbl@ifunset{bbl@\csname bbl@info@#1\endcsname @\languagename}%
4099     {\bbl@error{I've found no info for the current locale.\\%
4100                The corresponding ini file has not been loaded\\%
```

```
4101                    Perhaps it doesn't exist}%
4102                    {See the manual for details.}}%
4103     {\bbl@cs{\csname bbl@info@#1\endcsname @\languagename}}}
4104 % \@namedef{bbl@info@name.locale}{lcname}
4105 \@namedef{bbl@info@tag.ini}{lini}
4106 \@namedef{bbl@info@name.english}{elname}
4107 \@namedef{bbl@info@name.opentype}{lname}
4108 \@namedef{bbl@info@tag.bcp47}{tbcp}
4109 \@namedef{bbl@info@language.tag.bcp47}{lbcp}
4110 \@namedef{bbl@info@tag.opentype}{lotf}
4111 \@namedef{bbl@info@script.name}{esname}
4112 \@namedef{bbl@info@script.name.opentype}{sname}
4113 \@namedef{bbl@info@script.tag.bcp47}{sbcp}
4114 \@namedef{bbl@info@script.tag.opentype}{sotf}
4115 \let\bbl@ensureinfo\@gobble
4116 \newcommand\BabelEnsureInfo{%
4117    \ifx\InputIfFileExists\@undefined\else
4118      \def\bbl@ensureinfo##1{%
4119        \bbl@ifunset{bbl@lname@##1}{\bbl@load@info{##1}}{}}%
4120    \fi
4121    \bbl@foreach\bbl@loaded{{%
4122      \def\languagename{##1}%
4123      \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.

```
4124 \newcommand\getlocaleproperty{%
4125    \@ifstar\bbl@getproperty@s\bbl@getproperty@x}
4126 \def\bbl@getproperty@s#1#2#3{%
4127    \let#1\relax
4128    \def\bbl@elt##1##2##3{%
4129      \bbl@ifsamestring{##1/##2}{#3}%
4130        {\providecommand#1{##3}%
4131         \def\bbl@elt####1####2####3{}}%
4132        {}}%
4133    \bbl@cs{inidata@#2}}%
4134 \def\bbl@getproperty@x#1#2#3{%
4135    \bbl@getproperty@s{#1}{#2}{#3}%
4136    \ifx#1\relax
4137      \bbl@error
4138        {Unknown key for locale '#2':\\%
4139         #3\\%
4140         \string#1 will be set to \relax}%
4141        {Perhaps you misspelled it.}%
4142    \fi}
4143 \let\bbl@ini@loaded\@empty
4144 \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.

```
4145 \newcommand\babeladjust[1]{%  TODO. Error handling.
4146    \bbl@forkv{#1}{%
4147      \bbl@ifunset{bbl@ADJ@##1@##2}%
4148        {\bbl@cs{ADJ@##1}{##2}}%
4149        {\bbl@cs{ADJ@##1@##2}}}}
4150 %
```

```
4151 \def\bbl@adjust@lua#1#2{%
4152   \ifvmode
4153     \ifnum\currentgrouplevel=\z@
4154       \directlua{ Babel.#2 }%
4155       \expandafter\expandafter\expandafter\@gobble
4156     \fi
4157   \fi
4158   {\bbl@error   % The error is gobbled if everything went ok.
4159     {Currently, #1 related features can be adjusted only\\%
4160      in the main vertical list.}%
4161     {Maybe things change in the future, but this is what it is.}}}
4162 \@namedef{bbl@ADJ@bidi.mirroring@on}{%
4163   \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
4164 \@namedef{bbl@ADJ@bidi.mirroring@off}{%
4165   \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
4166 \@namedef{bbl@ADJ@bidi.text@on}{%
4167   \bbl@adjust@lua{bidi}{bidi_enabled=true}}
4168 \@namedef{bbl@ADJ@bidi.text@off}{%
4169   \bbl@adjust@lua{bidi}{bidi_enabled=false}}
4170 \@namedef{bbl@ADJ@bidi.mapdigits@on}{%
4171   \bbl@adjust@lua{bidi}{digits_mapped=true}}
4172 \@namedef{bbl@ADJ@bidi.mapdigits@off}{%
4173   \bbl@adjust@lua{bidi}{digits_mapped=false}}
4174 %
4175 \@namedef{bbl@ADJ@linebreak.sea@on}{%
4176   \bbl@adjust@lua{linebreak}{sea_enabled=true}}
4177 \@namedef{bbl@ADJ@linebreak.sea@off}{%
4178   \bbl@adjust@lua{linebreak}{sea_enabled=false}}
4179 \@namedef{bbl@ADJ@linebreak.cjk@on}{%
4180   \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
4181 \@namedef{bbl@ADJ@linebreak.cjk@off}{%
4182   \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
4183 \@namedef{bbl@ADJ@justify.arabic@on}{%
4184   \bbl@adjust@lua{linebreak}{arabic.justify_enabled=true}}
4185 \@namedef{bbl@ADJ@justify.arabic@off}{%
4186   \bbl@adjust@lua{linebreak}{arabic.justify_enabled=false}}
4187 %
4188 \def\bbl@adjust@layout#1{%
4189   \ifvmode
4190     #1%
4191     \expandafter\@gobble
4192   \fi
4193   {\bbl@error   % The error is gobbled if everything went ok.
4194     {Currently, layout related features can be adjusted only\\%
4195      in vertical mode.}%
4196     {Maybe things change in the future, but this is what it is.}}}
4197 \@namedef{bbl@ADJ@layout.tabular@on}{%
4198   \bbl@adjust@layout{\let\@tabular\bbl@NL@@tabular}}
4199 \@namedef{bbl@ADJ@layout.tabular@off}{%
4200   \bbl@adjust@layout{\let\@tabular\bbl@OL@@tabular}}
4201 \@namedef{bbl@ADJ@layout.lists@on}{%
4202   \bbl@adjust@layout{\let\list\bbl@NL@list}}
4203 \@namedef{bbl@ADJ@layout.lists@off}{%
4204   \bbl@adjust@layout{\let\list\bbl@OL@list}}
4205 \@namedef{bbl@ADJ@hyphenation.extra@on}{%
4206   \bbl@activateposthyphen}
4207 %
4208 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
4209   \bbl@bcpallowedtrue}
```

```
4210 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
4211   \bbl@bcpallowedfalse}
4212 \@namedef{bbl@ADJ@autoload.bcp47.prefix}#1{%
4213   \def\bbl@bcp@prefix{#1}}
4214 \def\bbl@bcp@prefix{bcp47-}
4215 \@namedef{bbl@ADJ@autoload.options}#1{%
4216   \def\bbl@autoload@options{#1}}
4217 \let\bbl@autoload@bcpoptions\@empty
4218 \@namedef{bbl@ADJ@autoload.bcp47.options}#1{%
4219   \def\bbl@autoload@bcpoptions{#1}}
4220 \newif\ifbbl@bcptoname
4221 \@namedef{bbl@ADJ@bcp47.toname@on}{%
4222   \bbl@bcptonametrue
4223   \BabelEnsureInfo}
4224 \@namedef{bbl@ADJ@bcp47.toname@off}{%
4225   \bbl@bcptonamefalse}
4226 \@namedef{bbl@ADJ@prehyphenation.disable@nohyphenation}{%
4227   \directlua{ Babel.ignore_pre_char = function(node)
4228       return (node.lang == \the\csname l@nohyphenation\endcsname)
4229     end }}
4230 \@namedef{bbl@ADJ@prehyphenation.disable@off}{%
4231   \directlua{ Babel.ignore_pre_char = function(node)
4232       return false
4233     end }}
4234 % TODO: use babel name, override
4235 %
4236 % As the final task, load the code for lua.
4237 %
4238 \ifx\directlua\@undefined\else
4239   \ifx\bbl@luapatterns\@undefined
4240     \input luababel.def
4241   \fi
4242 \fi
4243 ⟨/core⟩
```

A proxy file for switch.def

```
4244 ⟨*kernel⟩
4245 \let\bbl@onlyswitch\@empty
4246 \input babel.def
4247 \let\bbl@onlyswitch\@undefined
4248 ⟨/kernel⟩
4249 ⟨*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.

```
4250 ⟨⟨Make sure ProvidesFile is defined⟩⟩
4251 \ProvidesFile{hyphen.cfg}[⟨⟨date⟩⟩ ⟨⟨version⟩⟩ Babel hyphens]
4252 \xdef\bbl@format{\jobname}
```

```
4253 \def\bbl@version{⟨⟨version⟩⟩}
4254 \def\bbl@date{⟨⟨date⟩⟩}
4255 \ifx\AtBeginDocument\@undefined
4256   \def\@empty{}
4257   \let\orig@dump\dump
4258   \def\dump{%
4259     \ifx\@ztryfc\@undefined
4260     \else
4261       \toks0=\expandafter{\@preamblecmds}%
4262       \edef\@preamblecmds{\noexpand\@begindocumenthook\the\toks0}%
4263       \def\@begindocumenthook{}%
4264     \fi
4265     \let\dump\orig@dump\let\orig@dump\@undefined\dump}
4266 \fi
4267 ⟨⟨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.

```
4268 \def\process@line#1#2 #3 #4 {%
4269   \ifx=#1%
4270     \process@synonym{#2}%
4271   \else
4272     \process@language{#1#2}{#3}{#4}%
4273   \fi
4274   \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.

```
4275 \toks@{}
4276 \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.

```
4277 \def\process@synonym#1{%
4278   \ifnum\last@language=\m@ne
4279     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4280   \else
4281     \expandafter\chardef\csname l@#1\endcsname\last@language
4282     \wlog{\string\l@#1=\string\language\the\last@language}%
4283     \expandafter\let\csname #1hyphenmins\expandafter\endcsname
4284       \csname\languagename hyphenmins\endcsname
4285     \let\bbl@elt\relax
4286     \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}{}{}}%
4287   \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

158

\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.

```
4288 \def\process@language#1#2#3{%
4289   \expandafter\addlanguage\csname l@#1\endcsname
4290   \expandafter\language\csname l@#1\endcsname
4291   \edef\languagename{#1}%
4292   \bbl@hook@everylanguage{#1}%
4293   %  > luatex
4294   \bbl@get@enc#1::\@@@
4295   \begingroup
4296     \lefthyphenmin\m@ne
4297     \bbl@hook@loadpatterns{#2}%
4298     %  > luatex
4299     \ifnum\lefthyphenmin=\m@ne
4300     \else
4301       \expandafter\xdef\csname #1hyphenmins\endcsname{%
4302         \the\lefthyphenmin\the\righthyphenmin}%
4303     \fi
4304   \endgroup
4305   \def\bbl@tempa{#3}%
4306   \ifx\bbl@tempa\@empty\else
4307     \bbl@hook@loadexceptions{#3}%
4308     %  > luatex
4309   \fi
4310   \let\bbl@elt\relax
4311   \edef\bbl@languages{%
4312     \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
4313   \ifnum\the\language=\z@
4314     \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4315       \set@hyphenmins\tw@\thr@@\relax
4316     \else
4317       \expandafter\expandafter\expandafter\set@hyphenmins
4318         \csname #1hyphenmins\endcsname
4319     \fi
4320     \the\toks@
4321     \toks@{}%
4322   \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.

```
4323 \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.

```
4324 \def\bbl@hook@everylanguage#1{}
4325 \def\bbl@hook@loadpatterns#1{\input #1\relax}
4326 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
4327 \def\bbl@hook@loadkernel#1{%
4328   \def\addlanguage{\csname newlanguage\endcsname}%
4329   \def\adddialect##1##2{%
4330     \global\chardef##1##2\relax
4331     \wlog{\string##1 = a dialect from \string\language##2}}%
4332   \def\iflanguage##1{%
4333     \expandafter\ifx\csname l@##1\endcsname\relax
4334       \@nolanerr{##1}%
4335     \else
4336       \ifnum\csname l@##1\endcsname=\language
4337         \expandafter\expandafter\expandafter\@firstoftwo
4338       \else
4339         \expandafter\expandafter\expandafter\@secondoftwo
4340       \fi
4341     \fi}%
4342   \def\providehyphenmins##1##2{%
4343     \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
4344       \@namedef{##1hyphenmins}{##2}%
4345     \fi}%
4346   \def\set@hyphenmins##1##2{%
4347     \lefthyphenmin##1\relax
4348     \righthyphenmin##2\relax}%
4349   \def\selectlanguage{%
4350     \errhelp{Selecting a language requires a package supporting it}%
4351     \errmessage{Not loaded}}%
4352   \let\foreignlanguage\selectlanguage
4353   \let\otherlanguage\selectlanguage
4354   \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4355   \def\bbl@usehooks##1##2{}% TODO. Temporary!!
4356   \def\setlocale{%
4357     \errhelp{Find an armchair, sit down and wait}%
4358     \errmessage{Not yet available}}%
4359   \let\uselocale\setlocale
4360   \let\locale\setlocale
4361   \let\selectlocale\setlocale
4362   \let\localename\setlocale
4363   \let\textlocale\setlocale
4364   \let\textlanguage\setlocale
4365   \let\languagetext\setlocale}
4366 \begingroup
4367   \def\AddBabelHook#1#2{%
4368     \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
4369       \def\next{\toks1}%
4370     \else
4371       \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname####1}%
4372     \fi
4373     \next}
4374   \ifx\directlua\@undefined
4375     \ifx\XeTeXinputencoding\@undefined\else
4376       \input xebabel.def
4377     \fi
4378   \else
```

```
4379      \input luababel.def
4380   \fi
4381   \openin1 = babel-\bbl@format.cfg
4382   \ifeof1
4383   \else
4384      \input babel-\bbl@format.cfg\relax
4385   \fi
4386   \closein1
4387 \endgroup
4388 \bbl@hook@loadkernel{switch.def}
```

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

```
4389 \openin1 = language.dat
```

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

```
4390 \def\languagename{english}%
4391 \ifeof1
4392   \message{I couldn't find the file language.dat,\space
4393            I will try the file hyphen.tex}
4394   \input hyphen.tex\relax
4395   \chardef\l@english\z@
4396 \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.

```
4397   \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.

```
4398   \loop
4399     \endlinechar\m@ne
4400     \read1 to \bbl@line
4401     \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.

```
4402     \if T\ifeof1F\fi T\relax
4403       \ifx\bbl@line\@empty\else
4404         \edef\bbl@line{\bbl@line\space\space\space}%
4405         \expandafter\process@line\bbl@line\relax
4406       \fi
4407   \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.

```
4408   \begingroup
4409     \def\bbl@elt#1#2#3#4{%
4410       \global\language=#2\relax
4411       \gdef\languagename{#1}%
4412       \def\bbl@elt##1##2##3##4{}}%
4413     \bbl@languages
4414   \endgroup
4415 \fi
4416 \closein1
```

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

```
4417 \if/\the\toks@/\else
4418   \errhelp{language.dat loads no language, only synonyms}
4419   \errmessage{Orphan language synonym}
4420 \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.

```
4421 \let\bbl@line\@undefined
4422 \let\process@line\@undefined
4423 \let\process@synonym\@undefined
4424 \let\process@language\@undefined
4425 \let\bbl@get@enc\@undefined
4426 \let\bbl@hyph@enc\@undefined
4427 \let\bbl@tempa\@undefined
4428 \let\bbl@hook@loadkernel\@undefined
4429 \let\bbl@hook@everylanguage\@undefined
4430 \let\bbl@hook@loadpatterns\@undefined
4431 \let\bbl@hook@loadexceptions\@undefined
4432 ⟨/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].

```
4433 ⟨⟨∗More package options⟩⟩ ≡
4434 \chardef\bbl@bidimode\z@
4435 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4436 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
4437 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4438 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4439 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4440 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4441 ⟨⟨/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.

```
4442 ⟨⟨∗Font selection⟩⟩ ≡
4443 \bbl@trace{Font handling with fontspec}
4444 \ifx\ExplSyntaxOn\@undefined\else
4445   \ExplSyntaxOn
4446   \catcode`\ =10
4447   \def\bbl@loadfontspec{%
4448     \usepackage{fontspec}%
4449     \expandafter
4450     \def\csname msg~text~>~fontspec/language-not-exist\endcsname##1##2##3##4{%
4451       Font '\l_fontspec_fontname_tl' is using the\\%
4452       default features for language '##1'.\\%
4453       That's usually fine, because many languages\\%
4454       require no specific features, but if the output is\\%
4455       not as expected, consider selecting another font.}
```

```
4456      \expandafter
4457      \def\csname msg~text~>~fontspec/no-script\endcsname##1##2##3##4{%
4458        Font '\l_fontspec_fontname_tl' is using the\\%
4459        default features for script '##2'.\\%
4460        That's not always wrong, but if the output is\\%
4461        not as expected, consider selecting another font.}}
4462    \ExplSyntaxOff
4463 \fi
4464 \@onlypreamble\babelfont
4465 \newcommand\babelfont[2][]{%  1=langs/scripts 2=fam
4466    \bbl@foreach{#1}{%
4467      \expandafter\ifx\csname date##1\endcsname\relax
4468        \IfFileExists{babel-##1.tex}%
4469          {\babelprovide{##1}}%
4470          {}%
4471      \fi}%
4472    \edef\bbl@tempa{#1}%
4473    \def\bbl@tempb{#2}%  Used by \bbl@bblfont
4474    \ifx\fontspec\@undefined
4475      \bbl@loadfontspec
4476    \fi
4477    \EnableBabelHook{babel-fontspec}% Just calls \bbl@switchfont
4478    \bbl@bblfont}
4479 \newcommand\bbl@bblfont[2][]{% 1=features 2=fontname, @font=rm|sf|tt
4480    \bbl@ifunset{\bbl@tempb family}%
4481      {\bbl@providefam{\bbl@tempb}}%
4482      {\bbl@exp{%
4483        \\\bbl@sreplace\<\bbl@tempb family >%
4484          {\@nameuse{\bbl@tempb default}}{\<\bbl@tempb default>}}}%
4485    % For the default font, just in case:
4486    \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
4487    \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
4488      {\bbl@csarg\edef{\bbl@tempb dflt@}{<>{#1}{#2}}% save bbl@rmdflt@
4489        \bbl@exp{%
4490          \let\<bbl@\bbl@tempb dflt@\languagename>\<bbl@\bbl@tempb dflt@>%
4491          \\\bbl@font@set\<bbl@\bbl@tempb dflt@\languagename>%
4492                          \<\bbl@tempb default>\<\bbl@tempb family>}}%
4493      {\bbl@foreach\bbl@tempa{% ie bbl@rmdflt@lang / *scrt
4494          \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:

```
4495 \def\bbl@providefam#1{%
4496    \bbl@exp{%
4497      \\\newcommand\<#1default>{}% Just define it
4498      \\\bbl@add@list\\\bbl@font@fams{#1}%
4499      \\\DeclareRobustCommand\<#1family>{%
4500        \\\not@math@alphabet\<#1family>\relax
4501        \\\fontfamily\<#1default>\\\selectfont}%
4502      \\\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.

```
4503 \def\bbl@nostdfont#1{%
4504    \bbl@ifunset{bbl@WFF@\f@family}%
4505      {\bbl@csarg\gdef{WFF@\f@family}{}%  Flag, to avoid dupl warns
4506        \bbl@infowarn{The current font is not a babel standard family:\\%
4507          #1%
4508          \fontname\font\\%
4509          There is nothing intrinsically wrong with this warning, and\\%
```

```
4510        you can ignore it altogether if you do not need these\\%
4511        families. But if they are used in the document, you should be\\%
4512        aware 'babel' will no set Script and Language for them, so\\%
4513        you may consider defining a new family with \string\babelfont.\\%
4514        See the manual for further details about \string\babelfont.\\%
4515        Reported}}
4516    {}}%
4517 \gdef\bbl@switchfont{%
4518  \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
4519  \bbl@exp{%  eg Arabic -> arabic
4520    \lowercase{\edef\\\bbl@tempa{\bbl@cl{sname}}}}%
4521  \bbl@foreach\bbl@font@fams{%
4522    \bbl@ifunset{bbl@##1dflt@\languagename}%      (1) language?
4523      {\bbl@ifunset{bbl@##1dflt@*\bbl@tempa}%     (2) from script?
4524        {\bbl@ifunset{bbl@##1dflt@}%              2=F - (3) from generic?
4525          {}%                                    123=F - nothing!
4526          {\bbl@exp{%                            3=T - from generic
4527            \global\let\<bbl@##1dflt@\languagename>%
4528                      \<bbl@##1dflt@>}}}%
4529        {\bbl@exp{%                              2=T - from script
4530          \global\let\<bbl@##1dflt@\languagename>%
4531                    \<bbl@##1dflt@*\bbl@tempa>}}}%
4532      {}}%                                       1=T - language, already defined
4533  \def\bbl@tempa{\bbl@nostdfont{}}%
4534  \bbl@foreach\bbl@font@fams{%     don't gather with prev for
4535    \bbl@ifunset{bbl@##1dflt@\languagename}%
4536      {\bbl@cs{famrst@##1}%
4537      \global\bbl@csarg\let{famrst@##1}\relax}%
4538      {\bbl@exp{% order is relevant. TODO: but sometimes wrong!
4539        \\\bbl@add\\\originalTeX{%
4540          \\\bbl@font@rst{\bbl@cl{##1dflt}}%
4541                        \<##1default>\<##1family>{##1}}%
4542        \\\bbl@font@set\<bbl@##1dflt@\languagename>% the main part!
4543                      \<##1default>\<##1family>}}}%
4544  \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.

```
4545 \ifx\f@family\@undefined\else   % if latex
4546  \ifcase\bbl@engine            % if pdftex
4547    \let\bbl@ckeckstdfonts\relax
4548  \else
4549    \def\bbl@ckeckstdfonts{%
4550      \begingroup
4551        \global\let\bbl@ckeckstdfonts\relax
4552        \let\bbl@tempa\@empty
4553        \bbl@foreach\bbl@font@fams{%
4554          \bbl@ifunset{bbl@##1dflt@}%
4555            {\@nameuse{##1family}%
4556            \bbl@csarg\gdef{WFF@\f@family}{}% Flag
4557            \bbl@exp{\\\bbl@add\\\bbl@tempa{* \<##1family>= \f@family\\\\%
4558              \space\space\fontname\font\\\\}}%
4559            \bbl@csarg\xdef{##1dflt@}{\f@family}%
4560            \expandafter\xdef\csname ##1default\endcsname{\f@family}}%
4561            {}}%
4562        \ifx\bbl@tempa\@empty\else
4563          \bbl@infowarn{The following font families will use the default\\%
4564            settings for all or some languages:\\%
4565            \bbl@tempa
```

164

```
4566                There is nothing intrinsically wrong with it, but\\%
4567                'babel' will no set Script and Language, which could\\%
4568                 be relevant in some languages. If your document uses\\%
4569                 these families, consider redefining them with \string\babelfont.\\%
4570                Reported}%
4571          \fi
4572       \endgroup}
4573   \fi
4574 \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.

```
4575 \def\bbl@font@set#1#2#3{% eg \bbl@rmdflt@lang \rmdefault \rmfamily
4576   \bbl@xin@{<>}{#1}%
4577   \ifin@
4578     \bbl@exp{\\\bbl@fontspec@set\\#1\expandafter\@gobbletwo#1\\#3}%
4579   \fi
4580   \bbl@exp{%                'Unprotected' macros return prev values
4581     \def\\#2{#1}%          eg, \rmdefault{\bbl@rmdflt@lang}
4582     \\\bbl@ifsamestring{#2}{\f@family}%
4583       {\\#3%
4584        \\\bbl@ifsamestring{\f@series}{\bfdefault}{\\\bfseries}{}%
4585        \let\\\bbl@tempa\relax}%
4586       {}}}
4587 %     TODO - next should be global?, but even local does its job. I'm
4588 %     still not sure -- must investigate:
4589 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
4590   \let\bbl@tempe\bbl@mapselect
4591   \let\bbl@mapselect\relax
4592   \let\bbl@temp@fam#4%        eg, '\rmfamily', to be restored below
4593   \let#4\@empty     %        Make sure \renewfontfamily is valid
4594   \bbl@exp{%
4595     \let\\\bbl@temp@pfam\<\bbl@stripslash#4\space>% eg, '\rmfamily '
4596     \<keys_if_exist:nnF>{fontspec-opentype}{Script/\bbl@cl{sname}}%
4597       {\\\newfontscript{\bbl@cl{sname}}{\bbl@cl{sotf}}}%
4598     \<keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@cl{lname}}%
4599       {\\\newfontlanguage{\bbl@cl{lname}}{\bbl@cl{lotf}}}%
4600     \\\renewfontfamily\\#4%
4601       [\bbl@cs{lsys@\languagename},#2]}{#3}% ie \bbl@exp{..}{#3}
4602   \begingroup
4603     #4%
4604     \xdef#1{\f@family}%      eg, \bbl@rmdflt@lang{FreeSerif(0)}
4605   \endgroup
4606   \let#4\bbl@temp@fam
4607   \bbl@exp{\let\<\bbl@stripslash#4\space>}\bbl@temp@pfam
4608   \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.

```
4609 \def\bbl@font@rst#1#2#3#4{%
4610   \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.

```
4611 \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 :-).

```
4612 \newcommand\babelFSstore[2][]{%
4613   \bbl@ifblank{#1}%
4614     {\bbl@csarg\def{sname@#2}{Latin}}%
4615     {\bbl@csarg\def{sname@#2}{#1}}%
4616   \bbl@provide@dirs{#2}%
4617   \bbl@csarg\ifnum{wdir@#2}>\z@
4618     \let\bbl@beforeforeign\leavevmode
4619     \EnableBabelHook{babel-bidi}%
4620   \fi
4621   \bbl@foreach{#2}{%
4622     \bbl@FSstore{##1}{rm}\rmdefault\bbl@save@rmdefault
4623     \bbl@FSstore{##1}{sf}\sfdefault\bbl@save@sfdefault
4624     \bbl@FSstore{##1}{tt}\ttdefault\bbl@save@ttdefault}}
4625 \def\bbl@FSstore#1#2#3#4{%
4626   \bbl@csarg\edef{#2default#1}{#3}%
4627   \expandafter\addto\csname extras#1\endcsname{%
4628     \let#4#3%
4629     \ifx#3\f@family
4630       \edef#3{\csname bbl@#2default#1\endcsname}%
4631       \fontfamily{#3}\selectfont
4632     \else
4633       \edef#3{\csname bbl@#2default#1\endcsname}%
4634     \fi}%
4635   \expandafter\addto\csname noextras#1\endcsname{%
4636     \ifx#3\f@family
4637       \fontfamily{#4}\selectfont
4638     \fi
4639     \let#3#4}}
4640 \let\bbl@langfeatures\@empty
4641 \def\babelFSfeatures{% make sure \fontspec is redefined once
4642   \let\bbl@ori@fontspec\fontspec
4643   \renewcommand\fontspec[1][]{%
4644     \bbl@ori@fontspec[\bbl@langfeatures##1]}%
4645   \let\babelFSfeatures\bbl@FSfeatures
4646   \babelFSfeatures}
4647 \def\bbl@FSfeatures#1#2{%
4648   \expandafter\addto\csname extras#1\endcsname{%
4649     \babel@save\bbl@langfeatures
4650     \edef\bbl@langfeatures{#2,}}}
4651 ⟨⟨/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.

```
4652 ⟨⟨∗Footnote changes⟩⟩ ≡
4653 \bbl@trace{Bidi footnotes}
4654 \ifnum\bbl@bidimode>\z@
4655   \def\bbl@footnote#1#2#3{%
4656     \@ifnextchar[%
4657       {\bbl@footnote@o{#1}{#2}{#3}}%
4658       {\bbl@footnote@x{#1}{#2}{#3}}}
4659   \long\def\bbl@footnote@x#1#2#3#4{%
4660     \bgroup
4661       \select@language@x{\bbl@main@language}%
4662       \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
```

```
4663      \egroup}
4664   \long\def\bbl@footnote@o#1#2#3[#4]#5{%
4665      \bgroup
4666        \select@language@x{\bbl@main@language}%
4667        \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
4668      \egroup}
4669   \def\bbl@footnotetext#1#2#3{%
4670      \@ifnextchar[%
4671        {\bbl@footnotetext@o{#1}{#2}{#3}}%
4672        {\bbl@footnotetext@x{#1}{#2}{#3}}}
4673   \long\def\bbl@footnotetext@x#1#2#3#4{%
4674      \bgroup
4675        \select@language@x{\bbl@main@language}%
4676        \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
4677      \egroup}
4678   \long\def\bbl@footnotetext@o#1#2#3[#4]#5{%
4679      \bgroup
4680        \select@language@x{\bbl@main@language}%
4681        \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
4682      \egroup}
4683   \def\BabelFootnote#1#2#3#4{%
4684      \ifx\bbl@fn@footnote\@undefined
4685        \let\bbl@fn@footnote\footnote
4686      \fi
4687      \ifx\bbl@fn@footnotetext\@undefined
4688        \let\bbl@fn@footnotetext\footnotetext
4689      \fi
4690      \bbl@ifblank{#2}%
4691        {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
4692         \@namedef{\bbl@stripslash#1text}%
4693           {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
4694        {\def#1{\bbl@exp{\\\bbl@footnote{\\\foreignlanguage{#2}}}{#3}{#4}}%
4695         \@namedef{\bbl@stripslash#1text}%
4696           {\bbl@exp{\\\bbl@footnotetext{\\\foreignlanguage{#2}}}{#3}{#4}}}}
4697 \fi
4698 ⟨⟨/Footnote changes⟩⟩
```

Now, the code.

```
4699 ⟨∗xetex⟩
4700 \def\BabelStringsDefault{unicode}
4701 \let\xebbl@stop\relax
4702 \AddBabelHook{xetex}{encodedcommands}{%
4703   \def\bbl@tempa{#1}%
4704   \ifx\bbl@tempa\@empty
4705     \XeTeXinputencoding"bytes"%
4706   \else
4707     \XeTeXinputencoding"#1"%
4708   \fi
4709   \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4710 \AddBabelHook{xetex}{stopcommands}{%
4711   \xebbl@stop
4712   \let\xebbl@stop\relax}
4713 \def\bbl@intraspace#1 #2 #3\@@{%
4714   \bbl@csarg\gdef{xeisp@\languagename}%
4715     {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4716 \def\bbl@intrapenalty#1\@@{%
4717   \bbl@csarg\gdef{xeipn@\languagename}%
4718     {\XeTeXlinebreakpenalty #1\relax}}
4719 \def\bbl@provide@intraspace{%
```

```
4720    \bbl@xin@{/s}{/\bbl@cl{lnbrk}}%
4721    \ifin@\else\bbl@xin@{/c}{/\bbl@cl{lnbrk}}\fi
4722    \ifin@
4723      \bbl@ifunset{bbl@intsp@\languagename}{}%
4724        {\expandafter\ifx\csname bbl@intsp@\languagename\endcsname\@empty\else
4725          \ifx\bbl@KVP@intraspace\@nil
4726            \bbl@exp{%
4727              \\\bbl@intraspace\bbl@cl{intsp}\\\@@}%
4728          \fi
4729          \ifx\bbl@KVP@intrapenalty\@nil
4730            \bbl@intrapenalty0\@@
4731          \fi
4732        \fi
4733        \ifx\bbl@KVP@intraspace\@nil\else % We may override the ini
4734          \expandafter\bbl@intraspace\bbl@KVP@intraspace\@@
4735        \fi
4736        \ifx\bbl@KVP@intrapenalty\@nil\else
4737          \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
4738        \fi
4739        \bbl@exp{%
4740          \\\bbl@add\<extras\languagename>{%
4741            \XeTeXlinebreaklocale "\bbl@cl{tbcp}"%
4742            \<bbl@xeisp@\languagename>%
4743            \<bbl@xeipn@\languagename>}%
4744          \\\bbl@toglobal\<extras\languagename>%
4745          \\\bbl@add\<noextras\languagename>{%
4746            \XeTeXlinebreaklocale "en"}%
4747          \\\bbl@toglobal\<noextras\languagename>}%
4748        \ifx\bbl@ispacesize\@undefined
4749          \gdef\bbl@ispacesize{\bbl@cl{xeisp}}%
4750          \ifx\AtBeginDocument\@notprerr
4751            \expandafter\@secondoftwo  % to execute right now
4752          \fi
4753          \AtBeginDocument{%
4754            \expandafter\bbl@add
4755            \csname selectfont \endcsname{\bbl@ispacesize}%
4756            \expandafter\bbl@toglobal\csname selectfont \endcsname}%
4757        \fi}%
4758    \fi}
4759  \ifx\DisableBabelHook\@undefined\endinput\fi
4760  \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4761  \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
4762  \DisableBabelHook{babel-fontspec}
4763  ⟨⟨Font selection⟩⟩
4764  \input txtbabel.def
4765  ⟨/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.

```
4766  ⟨*texxet⟩
4767  \providecommand\bbl@provide@intraspace{}
```

```
4768 \bbl@trace{Redefinitions for bidi layout}
4769 \def\bbl@sspre@caption{%
4770   \bbl@exp{\everyhbox{\\\bbl@textdir\bbl@cs{wdir@\bbl@main@language}}}}}
4771 \ifx\bbl@opt@layout\@nnil\endinput\fi  % No layout
4772 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
4773 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
4774 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
4775   \def\@hangfrom#1{%
4776     \setbox\@tempboxa\hbox{{#1}}%
4777     \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
4778     \noindent\box\@tempboxa}
4779   \def\raggedright{%
4780     \let\\\@centercr
4781     \bbl@startskip\z@skip
4782     \@rightskip\@flushglue
4783     \bbl@endskip\@rightskip
4784     \parindent\z@
4785     \parfillskip\bbl@startskip}
4786   \def\raggedleft{%
4787     \let\\\@centercr
4788     \bbl@startskip\@flushglue
4789     \bbl@endskip\z@skip
4790     \parindent\z@
4791     \parfillskip\bbl@endskip}
4792 \fi
4793 \IfBabelLayout{lists}
4794   {\bbl@sreplace\list
4795     {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
4796   \def\bbl@listleftmargin{%
4797     \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
4798   \ifcase\bbl@engine
4799     \def\labelenumii{)\theenumii(}% pdftex doesn't reverse ()
4800     \def\p@enumiii{\p@enumii)\theenumii(}%
4801   \fi
4802   \bbl@sreplace\@verbatim
4803     {\leftskip\@totalleftmargin}%
4804     {\bbl@startskip\textwidth
4805      \advance\bbl@startskip-\linewidth}%
4806   \bbl@sreplace\@verbatim
4807     {\rightskip\z@skip}%
4808     {\bbl@endskip\z@skip}}%
4809   {}
4810 \IfBabelLayout{contents}
4811   {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
4812   \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
4813   {}
4814 \IfBabelLayout{columns}
4815   {\bbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbl@outputhbox}%
4816   \def\bbl@outputhbox#1{%
4817     \hb@xt@\textwidth{%
4818       \hskip\columnwidth
4819       \hfil
4820       {\normalcolor\vrule \@width\columnseprule}%
4821       \hfil
4822       \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
4823       \hskip-\textwidth
4824       \hb@xt@\columnwidth{\box\@outputbox \hss}%
4825       \hskip\columnsep
4826       \hskip\columnwidth}}}%
```

```
4827    {}
```
4828 ⟨⟨*Footnote changes*⟩⟩
```
4829 \IfBabelLayout{footnotes}%
4830   {\BabelFootnote\footnote\languagename{}{}%
4831    \BabelFootnote\localfootnote\languagename{}{}%
4832    \BabelFootnote\mainfootnote{}{}{}}
4833   {}
```

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.

```
4834 \IfBabelLayout{counters}%
4835   {\let\bbl@latinarabic=\@arabic
4836    \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
4837    \let\bbl@asciiroman=\@roman
4838    \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciiroman#1}}}%
4839    \let\bbl@asciiRoman=\@Roman
4840    \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}}{}
```
4841 ⟨/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`).

4842 ⟨*luatex⟩
```
4843 \ifx\AddBabelHook\@undefined % When plain.def, babel.sty starts
4844 \bbl@trace{Read language.dat}
4845 \ifx\bbl@readstream\@undefined
4846   \csname newread\endcsname\bbl@readstream
```

```
4847 \fi
4848 \begingroup
4849   \toks@{}
4850   \count@\z@ % 0=start, 1=0th, 2=normal
4851   \def\bbl@process@line#1#2 #3 #4 {%
4852     \ifx=#1%
4853       \bbl@process@synonym{#2}%
4854     \else
4855       \bbl@process@language{#1#2}{#3}{#4}%
4856     \fi
4857     \ignorespaces}
4858   \def\bbl@manylang{%
4859     \ifnum\bbl@last>\@ne
4860       \bbl@info{Non-standard hyphenation setup}%
4861     \fi
4862     \let\bbl@manylang\relax}
4863   \def\bbl@process@language#1#2#3{%
4864     \ifcase\count@
4865       \@ifundefined{zth@#1}{\count@\tw@}{\count@\@ne}%
4866     \or
4867       \count@\tw@
4868     \fi
4869     \ifnum\count@=\tw@
4870       \expandafter\addlanguage\csname l@#1\endcsname
4871       \language\allocationnumber
4872       \chardef\bbl@last\allocationnumber
4873       \bbl@manylang
4874       \let\bbl@elt\relax
4875       \xdef\bbl@languages{%
4876         \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
4877     \fi
4878     \the\toks@
4879     \toks@{}}
4880   \def\bbl@process@synonym@aux#1#2{%
4881     \global\expandafter\chardef\csname l@#1\endcsname#2\relax
4882     \let\bbl@elt\relax
4883     \xdef\bbl@languages{%
4884       \bbl@languages\bbl@elt{#1}{#2}{}{}}}%
4885   \def\bbl@process@synonym#1{%
4886     \ifcase\count@
4887       \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
4888     \or
4889       \@ifundefined{zth@#1}{\bbl@process@synonym@aux{#1}{0}}{}%
4890     \else
4891       \bbl@process@synonym@aux{#1}{\the\bbl@last}%
4892     \fi}
4893 \ifx\bbl@languages\@undefined % Just a (sensible?) guess
4894   \chardef\l@english\z@
4895   \chardef\l@USenglish\z@
4896   \chardef\bbl@last\z@
4897   \global\@namedef{bbl@hyphendata@0}{{hyphen.tex}{}}
4898   \gdef\bbl@languages{%
4899     \bbl@elt{english}{0}{hyphen.tex}{}%
4900     \bbl@elt{USenglish}{0}{}{}}
4901 \else
4902   \global\let\bbl@languages@format\bbl@languages
4903   \def\bbl@elt#1#2#3#4{% Remove all except language 0
4904     \ifnum#2>\z@\else
4905       \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
```

171

```
4906        \fi}%
4907      \xdef\bbl@languages{\bbl@languages}%
4908    \fi
4909    \def\bbl@elt#1#2#3#4{\@namedef{zth@#1}{}} % Define flags
4910    \bbl@languages
4911    \openin\bbl@readstream=language.dat
4912    \ifeof\bbl@readstream
4913      \bbl@warning{I couldn't find language.dat. No additional\\%
4914                    patterns loaded. Reported}%
4915    \else
4916      \loop
4917        \endlinechar\m@ne
4918        \read\bbl@readstream to \bbl@line
4919        \endlinechar`\^^M
4920        \if T\ifeof\bbl@readstream F\fi T\relax
4921          \ifx\bbl@line\@empty\else
4922            \edef\bbl@line{\bbl@line\space\space\space}%
4923            \expandafter\bbl@process@line\bbl@line\relax
4924          \fi
4925      \repeat
4926    \fi
4927 \endgroup
4928 \bbl@trace{Macros for reading patterns files}
4929 \def\bbl@get@enc#1:#2:#3\@@@{\def\bbl@hyph@enc{#2}}
4930 \ifx\babelcatcodetablenum\@undefined
4931    \ifx\newcatcodetable\@undefined
4932      \def\babelcatcodetablenum{5211}
4933      \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4934    \else
4935      \newcatcodetable\babelcatcodetablenum
4936      \newcatcodetable\bbl@pattcodes
4937    \fi
4938 \else
4939    \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4940 \fi
4941 \def\bbl@luapatterns#1#2{%
4942    \bbl@get@enc#1::\@@@
4943    \setbox\z@\hbox\bgroup
4944      \begingroup
4945        \savecatcodetable\babelcatcodetablenum\relax
4946        \initcatcodetable\bbl@pattcodes\relax
4947        \catcodetable\bbl@pattcodes\relax
4948          \catcode`\#=6  \catcode`\$=3 \catcode`\&=4 \catcode`\^=7
4949          \catcode`\_=8  \catcode`\{=1 \catcode`\}=2 \catcode`\~=13
4950          \catcode`\@=11 \catcode`\^^I=10 \catcode`\^^J=12
4951          \catcode`\<=12 \catcode`\>=12 \catcode`\*=12 \catcode`\.=12
4952          \catcode`\-=12 \catcode`\/=12 \catcode`\[=12 \catcode`\]=12
4953          \catcode`\`=12 \catcode`\'=12 \catcode`\"=12
4954        \input #1\relax
4955        \catcodetable\babelcatcodetablenum\relax
4956      \endgroup
4957      \def\bbl@tempa{#2}%
4958      \ifx\bbl@tempa\@empty\else
4959        \input #2\relax
4960      \fi
4961    \egroup}%
4962 \def\bbl@patterns@lua#1{%
4963    \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
4964      \csname l@#1\endcsname
```

```
4965     \edef\bbl@tempa{#1}%
4966   \else
4967     \csname l@#1:\f@encoding\endcsname
4968     \edef\bbl@tempa{#1:\f@encoding}%
4969   \fi\relax
4970   \@namedef{lu@texhyphen@loaded@\the\language}{}% Temp
4971   \@ifundefined{bbl@hyphendata@\the\language}%
4972     {\def\bbl@elt##1##2##3##4{%
4973        \ifnum##2=\csname l@\bbl@tempa\endcsname % #2=spanish, dutch:OT1...
4974          \def\bbl@tempb{##3}%
4975          \ifx\bbl@tempb\@empty\else % if not a synonymous
4976            \def\bbl@tempc{{##3}{##4}}%
4977          \fi
4978          \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
4979        \fi}%
4980      \bbl@languages
4981      \@ifundefined{bbl@hyphendata@\the\language}%
4982        {\bbl@info{No hyphenation patterns were set for\\%
4983                  language '\bbl@tempa'. Reported}}%
4984        {\expandafter\expandafter\expandafter\bbl@luapatterns
4985          \csname bbl@hyphendata@\the\language\endcsname}}{}}
4986 \endinput\fi
4987   % Here ends \ifx\AddBabelHook\@undefined
4988   % A few lines are only read by hyphen.cfg
4989 \ifx\DisableBabelHook\@undefined
4990   \AddBabelHook{luatex}{everylanguage}{%
4991     \def\process@language##1##2##3{%
4992       \def\process@line####1####2 ####3 ####4 {}}}
4993   \AddBabelHook{luatex}{loadpatterns}{%
4994     \input #1\relax
4995     \expandafter\gdef\csname bbl@hyphendata@\the\language\endcsname
4996       {{#1}{}}}
4997   \AddBabelHook{luatex}{loadexceptions}{%
4998     \input #1\relax
4999     \def\bbl@tempb##1##2{{##1}{#1}}%
5000     \expandafter\xdef\csname bbl@hyphendata@\the\language\endcsname
5001       {\expandafter\expandafter\expandafter\bbl@tempb
5002        \csname bbl@hyphendata@\the\language\endcsname}}
5003 \endinput\fi
5004   % Here stops reading code for hyphen.cfg
5005   % The following is read the 2nd time it's loaded
5006 \begingroup  % TODO - to a lua file
5007 \catcode`\%=12
5008 \catcode`\'=12
5009 \catcode`\"=12
5010 \catcode`\:=12
5011 \directlua{
5012 Babel = Babel or {}
5013 function Babel.bytes(line)
5014   return line:gsub("(.)",
5015     function (chr) return unicode.utf8.char(string.byte(chr)) end)
5016 end
5017 function Babel.begin_process_input()
5018   if luatexbase and luatexbase.add_to_callback then
5019     luatexbase.add_to_callback('process_input_buffer',
5020                               Babel.bytes,'Babel.bytes')
5021   else
5022     Babel.callback = callback.find('process_input_buffer')
5023     callback.register('process_input_buffer',Babel.bytes)
```

```
5024        end
5025      end
5026      function Babel.end_process_input ()
5027        if luatexbase and luatexbase.remove_from_callback then
5028          luatexbase.remove_from_callback('process_input_buffer','Babel.bytes')
5029        else
5030          callback.register('process_input_buffer',Babel.callback)
5031        end
5032      end
5033      function Babel.addpatterns(pp, lg)
5034        local lg = lang.new(lg)
5035        local pats = lang.patterns(lg) or ''
5036        lang.clear_patterns(lg)
5037        for p in pp:gmatch('[^%s]+') do
5038          ss = ''
5039          for i in string.utfcharacters(p:gsub('%d', '')) do
5040            ss = ss .. '%d?' .. i
5041          end
5042          ss = ss:gsub('^%%d%?%.', '%%.') .. '%d?'
5043          ss = ss:gsub('%.%%d%?$', '%%.')
5044          pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
5045          if n == 0 then
5046            tex.sprint(
5047              [[\string\csname\space bbl@info\endcsname{New pattern: ]]
5048              .. p .. [[}]])
5049            pats = pats .. ' ' .. p
5050          else
5051            tex.sprint(
5052              [[\string\csname\space bbl@info\endcsname{Renew pattern: ]]
5053              .. p .. [[}]])
5054          end
5055        end
5056        lang.patterns(lg, pats)
5057      end
5058  }
5059  \endgroup
5060  \ifx\newattribute\@undefined\else
5061    \newattribute\bbl@attr@locale
5062    \directlua{ Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale'}
5063    \AddBabelHook{luatex}{beforeextras}{%
5064      \setattribute\bbl@attr@locale\localeid}
5065  \fi
5066  \def\BabelStringsDefault{unicode}
5067  \let\luabbl@stop\relax
5068  \AddBabelHook{luatex}{encodedcommands}{%
5069    \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
5070    \ifx\bbl@tempa\bbl@tempb\else
5071      \directlua{Babel.begin_process_input()}%
5072      \def\luabbl@stop{%
5073        \directlua{Babel.end_process_input()}}%
5074    \fi}%
5075  \AddBabelHook{luatex}{stopcommands}{%
5076    \luabbl@stop
5077    \let\luabbl@stop\relax}
5078  \AddBabelHook{luatex}{patterns}{%
5079    \@ifundefined{bbl@hyphendata@\the\language}%
5080      {\def\bbl@elt##1##2##3##4{%
5081        \ifnum##2=\csname l@##2\endcsname % #2=spanish, dutch:OT1...
5082          \def\bbl@tempb{##3}%
```

174

```
5083              \ifx\bbl@tempb\@empty\else % if not a synonymous
5084                \def\bbl@tempc{{##3}{##4}}%
5085              \fi
5086              \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5087            \fi}%
5088          \bbl@languages
5089          \@ifundefined{bbl@hyphendata@\the\language}%
5090            {\bbl@info{No hyphenation patterns were set for\\%
5091                       language '#2'. Reported}}%
5092            {\expandafter\expandafter\expandafter\bbl@luapatterns
5093              \csname bbl@hyphendata@\the\language\endcsname}}{}%
5094    \@ifundefined{bbl@patterns@}{}{%
5095      \begingroup
5096        \bbl@xin@{,\number\language,}{,\bbl@pttnlist}%
5097        \ifin@\else
5098          \ifx\bbl@patterns@\@empty\else
5099            \directlua{ Babel.addpatterns(
5100              [[\bbl@patterns@]], \number\language) }%
5101          \fi
5102          \@ifundefined{bbl@patterns@#1}%
5103            \@empty
5104            {\directlua{ Babel.addpatterns(
5105                [[\space\csname bbl@patterns@#1\endcsname]],
5106                \number\language) }}%
5107          \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
5108        \fi
5109      \endgroup}%
5110    \bbl@exp{%
5111      \bbl@ifunset{bbl@prehc@\languagename}{}%
5112        {\\\bbl@ifblank{\bbl@cs{prehc@\languagename}}{}%
5113          {\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.

```
5114 \@onlypreamble\babelpatterns
5115 \AtEndOfPackage{%
5116   \newcommand\babelpatterns[2][\@empty]{%
5117     \ifx\bbl@patterns@\relax
5118       \let\bbl@patterns@\@empty
5119     \fi
5120     \ifx\bbl@pttnlist\@empty\else
5121       \bbl@warning{%
5122         You must not intermingle \string\selectlanguage\space and\\%
5123         \string\babelpatterns\space or some patterns will not\\%
5124         be taken into account. Reported}%
5125     \fi
5126     \ifx\@empty#1%
5127       \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
5128     \else
5129       \edef\bbl@tempb{\zap@space#1 \@empty}%
5130       \bbl@for\bbl@tempa\bbl@tempb{%
5131         \bbl@fixname\bbl@tempa
5132         \bbl@iflanguage\bbl@tempa{%
5133           \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
5134             \@ifundefined{bbl@patterns@\bbl@tempa}%
5135               \@empty
5136               {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
5137             #2}}}%
```

```
5138    \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.

```
5139 % TODO - to a lua file
5140 \directlua{
5141   Babel = Babel or {}
5142   Babel.linebreaking = Babel.linebreaking or {}
5143   Babel.linebreaking.before = {}
5144   Babel.linebreaking.after = {}
5145   Babel.locale = {} % Free to use, indexed by \localeid
5146   function Babel.linebreaking.add_before(func)
5147     tex.print([[\noexpand\csname bbl@luahyphenate\endcsname]])
5148     table.insert(Babel.linebreaking.before, func)
5149   end
5150   function Babel.linebreaking.add_after(func)
5151     tex.print([[\noexpand\csname bbl@luahyphenate\endcsname]])
5152     table.insert(Babel.linebreaking.after, func)
5153   end
5154 }
5155 \def\bbl@intraspace#1 #2 #3\@@{%
5156   \directlua{
5157     Babel = Babel or {}
5158     Babel.intraspaces = Babel.intraspaces or {}
5159     Babel.intraspaces['\csname bbl@sbcp@\languagename\endcsname'] = %
5160       {b = #1, p = #2, m = #3}
5161     Babel.locale_props[\the\localeid].intraspace = %
5162       {b = #1, p = #2, m = #3}
5163   }}
5164 \def\bbl@intrapenalty#1\@@{%
5165   \directlua{
5166     Babel = Babel or {}
5167     Babel.intrapenalties = Babel.intrapenalties or {}
5168     Babel.intrapenalties['\csname bbl@sbcp@\languagename\endcsname'] = #1
5169     Babel.locale_props[\the\localeid].intrapenalty = #1
5170   }}
5171 \begingroup
5172 \catcode`\%=12
5173 \catcode`\^=14
5174 \catcode`\'=12
5175 \catcode`\~=12
5176 \gdef\bbl@seaintraspace{^
5177   \let\bbl@seaintraspace\relax
5178   \directlua{
5179     Babel = Babel or {}
5180     Babel.sea_enabled = true
5181     Babel.sea_ranges = Babel.sea_ranges or {}
5182     function Babel.set_chranges (script, chrng)
5183       local c = 0
5184       for s, e in string.gmatch(chrng..' ', '(.-)%.%.(.-)%s') do
5185         Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
5186         c = c + 1
5187       end
5188     end
```

```
5189    function Babel.sea_disc_to_space (head)
5190      local sea_ranges = Babel.sea_ranges
5191      local last_char = nil
5192      local quad = 655360        ^% 10 pt = 655360 = 10 * 65536
5193      for item in node.traverse(head) do
5194        local i = item.id
5195        if i == node.id'glyph' then
5196          last_char = item
5197        elseif i == 7 and item.subtype == 3 and last_char
5198            and last_char.char > 0x0C99 then
5199          quad = font.getfont(last_char.font).size
5200          for lg, rg in pairs(sea_ranges) do
5201            if last_char.char > rg[1] and last_char.char < rg[2] then
5202              lg = lg:sub(1, 4)  ^% Remove trailing number of, eg, Cyrl1
5203              local intraspace = Babel.intraspaces[lg]
5204              local intrapenalty = Babel.intrapenalties[lg]
5205              local n
5206              if intrapenalty ~= 0 then
5207                n = node.new(14, 0)      ^% penalty
5208                n.penalty = intrapenalty
5209                node.insert_before(head, item, n)
5210              end
5211              n = node.new(12, 13)        ^% (glue, spaceskip)
5212              node.setglue(n, intraspace.b * quad,
5213                              intraspace.p * quad,
5214                              intraspace.m * quad)
5215              node.insert_before(head, item, n)
5216              node.remove(head, item)
5217            end
5218          end
5219        end
5220      end
5221    end
5222  }^^
5223  \bbl@luahyphenate}
```

13.5   CJK line breaking

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

```
5224 \catcode`\%=14
5225 \gdef\bbl@cjkintraspace{%
5226  \let\bbl@cjkintraspace\relax
5227  \directlua{
5228    Babel = Babel or {}
5229    require('babel-data-cjk.lua')
5230    Babel.cjk_enabled = true
5231    function Babel.cjk_linebreak(head)
5232      local GLYPH = node.id'glyph'
5233      local last_char = nil
5234      local quad = 655360        % 10 pt = 655360 = 10 * 65536
5235      local last_class = nil
5236      local last_lang = nil
5237
```

```
5238      for item in node.traverse(head) do
5239        if item.id == GLYPH then
5240
5241          local lang = item.lang
5242
5243          local LOCALE = node.get_attribute(item,
5244                luatexbase.registernumber'bbl@attr@locale')
5245          local props = Babel.locale_props[LOCALE]
5246
5247          local class = Babel.cjk_class[item.char].c
5248
5249          if class == 'cp' then class = 'cl' end % )] as CL
5250          if class == 'id' then class = 'I' end
5251
5252          local br = 0
5253          if class and last_class and Babel.cjk_breaks[last_class][class] then
5254            br = Babel.cjk_breaks[last_class][class]
5255          end
5256
5257          if br == 1 and props.linebreak == 'c' and
5258              lang ~= \the\l@nohyphenation\space and
5259              last_lang ~= \the\l@nohyphenation then
5260            local intrapenalty = props.intrapenalty
5261            if intrapenalty ~= 0 then
5262              local n = node.new(14, 0)      % penalty
5263              n.penalty = intrapenalty
5264              node.insert_before(head, item, n)
5265            end
5266            local intraspace = props.intraspace
5267            local n = node.new(12, 13)      % (glue, spaceskip)
5268            node.setglue(n, intraspace.b * quad,
5269                            intraspace.p * quad,
5270                            intraspace.m * quad)
5271            node.insert_before(head, item, n)
5272          end
5273
5274          if font.getfont(item.font) then
5275            quad = font.getfont(item.font).size
5276          end
5277          last_class = class
5278          last_lang = lang
5279        else % if penalty, glue or anything else
5280          last_class = nil
5281        end
5282      end
5283      lang.hyphenate(head)
5284    end
5285  }%
5286  \bbl@luahyphenate}
5287 \gdef\bbl@luahyphenate{%
5288  \let\bbl@luahyphenate\relax
5289  \directlua{
5290    luatexbase.add_to_callback('hyphenate',
5291    function (head, tail)
5292      if Babel.linebreaking.before then
5293        for k, func in ipairs(Babel.linebreaking.before)  do
5294          func(head)
5295        end
5296      end
```

```
5297        if Babel.cjk_enabled then
5298          Babel.cjk_linebreak(head)
5299        end
5300        lang.hyphenate(head)
5301        if Babel.linebreaking.after then
5302          for k, func in ipairs(Babel.linebreaking.after)  do
5303            func(head)
5304          end
5305        end
5306        if Babel.sea_enabled then
5307          Babel.sea_disc_to_space(head)
5308        end
5309      end,
5310      'Babel.hyphenate')
5311  }
5312 }
5313 \endgroup
5314 \def\bbl@provide@intraspace{%
5315   \bbl@ifunset{bbl@intsp@\languagename}{}%
5316     {\expandafter\ifx\csname bbl@intsp@\languagename\endcsname\@empty\else
5317       \bbl@xin@{/c}{/\bbl@cl{lnbrk}}%
5318       \ifin@          % cjk
5319         \bbl@cjkintraspace
5320         \directlua{
5321            Babel = Babel or {}
5322            Babel.locale_props = Babel.locale_props or {}
5323            Babel.locale_props[\the\localeid].linebreak = 'c'
5324         }%
5325         \bbl@exp{\\\bbl@intraspace\bbl@cl{intsp}\\\@@}%
5326         \ifx\bbl@KVP@intrapenalty\@nil
5327           \bbl@intrapenalty0\@@
5328         \fi
5329       \else            % sea
5330         \bbl@seaintraspace
5331         \bbl@exp{\\\bbl@intraspace\bbl@cl{intsp}\\\@@}%
5332         \directlua{
5333            Babel = Babel or {}
5334            Babel.sea_ranges = Babel.sea_ranges or {}
5335            Babel.set_chranges('\bbl@cl{sbcp}',
5336                               '\bbl@cl{chrng}')
5337         }%
5338         \ifx\bbl@KVP@intrapenalty\@nil
5339           \bbl@intrapenalty0\@@
5340         \fi
5341       \fi
5342     \fi
5343     \ifx\bbl@KVP@intrapenalty\@nil\else
5344       \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
5345     \fi}}
```

13.6   Arabic justification

```
5346 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5347 \def\bblar@chars{%
5348   0628,0629,062A,062B,062C,062D,062E,062F,0630,0631,0632,0633,%
5349   0634,0635,0636,0637,0638,0639,063A,063B,063C,063D,063E,063F,%
5350   0640,0641,0642,0643,0644,0645,0646,0647,0649}
5351 \def\bblar@elongated{%
5352   0626,0628,062A,062B,0633,0634,0635,0636,063B,%
```

```
5353   063C,063D,063E,063F,0641,0642,0643,0644,0646,%
5354   0649,064A}
5355 \begingroup
5356   \catcode`\_=11 \catcode`\:=11
5357   \gdef\bblar@nofswarn{\gdef\msg_warning:nnx##1##2##3{}}
5358 \endgroup
5359 \gdef\bbl@arabicjust{%
5360   \let\bbl@arabicjust\relax
5361   \newattribute\bblar@kashida
5362   \bblar@kashida=\z@
5363   \expandafter\bbl@add\csname selectfont \endcsname{{\bbl@parsejalt}}%
5364   \directlua{
5365     Babel.arabic.elong_map   = Babel.arabic.elong_map or {}
5366     Babel.arabic.elong_map[\the\localeid]   = {}
5367     luatexbase.add_to_callback('post_linebreak_filter',
5368       Babel.arabic.justify, 'Babel.arabic.justify')
5369   }}%
5370 % Save both node lists to make replacement. TODO. Save also widths to
5371 % make computations
5372 \def\bblar@fetchjalt#1#2#3#4{%
5373   \bbl@exp{\\\bbl@foreach{#1}}{%
5374     \bbl@ifunset{bblar@JE@##1}%
5375       {\setbox\z@\hbox{^^^^200d\char"##1#2}}%
5376       {\setbox\z@\hbox{^^^^200d\char"\@nameuse{bblar@JE@##1}#2}}%
5377     \directlua{%
5378       local last = nil
5379       for item in node.traverse(tex.box[0].head) do
5380         if item.id == node.id'glyph' and item.char > 0x600 and
5381             not (item.char == 0x200D) then
5382           last = item
5383         end
5384       end
5385       Babel.arabic.#3['##1#4'] = last.char
5386     }}}
5387 % Brute force. No rules at all, yet. The ideal: look at jalt table. And
5388 % perhaps other tables (falt?, cswh?). What about kaf? And diacritic
5389 % positioning?
5390 \gdef\bbl@parsejalt{%
5391   \ifx\addfontfeature\@undefined\else
5392   \bbl@xin@{/e}{/\bbl@cl{lnbrk}}%
5393   \ifin@
5394     \directlua{%
5395       if Babel.arabic.elong_map[\the\localeid][\fontid\font] == nil then
5396         Babel.arabic.elong_map[\the\localeid][\fontid\font] = {}
5397         tex.print([[\string\csname\space bbl@parsejalti\endcsname]])
5398       end
5399     }%
5400   \fi
5401   \fi}
5402 \gdef\bbl@parsejalti{%
5403   \begingroup
5404     \let\bbl@parsejalt\relax     % To avoid infinite loop
5405     \edef\bbl@tempb{\fontid\font}%
5406     \bblar@nofswarn
5407     \bblar@fetchjalt\bblar@elongated{}{from}{}%
5408     \bblar@fetchjalt\bblar@chars{^^^^064a}{from}{a}% Alef maksura
5409     \bblar@fetchjalt\bblar@chars{^^^^0649}{from}{y}% Yeh
5410     \addfontfeature{RawFeature=+jalt}%
5411     % \@namedef{bblar@JE@0643}{06AA}% todo: catch medial kaf
```

```
5412      \bblar@fetchjalt\bblar@elongated{}{dest}{}%
5413      \bblar@fetchjalt\bblar@chars{^^^^064a}{dest}{a}%
5414      \bblar@fetchjalt\bblar@chars{^^^^0649}{dest}{y}%
5415       \directlua{%
5416         for k, v in pairs(Babel.arabic.from) do
5417           if Babel.arabic.dest[k] and
5418               not (Babel.arabic.from[k] == Babel.arabic.dest[k]) then
5419             Babel.arabic.elong_map[\the\localeid][\bbl@tempb]
5420               [Babel.arabic.from[k]] = Babel.arabic.dest[k]
5421           end
5422         end
5423       }%
5424   \endgroup}
5425 %
5426 \begingroup
5427 \catcode`\#=11
5428 \catcode`\~=11
5429 \directlua{
5430
5431 Babel.arabic = Babel.arabic or {}
5432 Babel.arabic.from = {}
5433 Babel.arabic.dest = {}
5434 Babel.arabic.justify_factor = 0.95
5435 Babel.arabic.justify_enabled = true
5436
5437 function Babel.arabic.justify(head)
5438   if not Babel.arabic.justify_enabled then return head end
5439   local d, new
5440   local k_list, k_item, pos_inline
5441   local width, width_new, full, k_curr, wt_pos, goal, shift
5442   local subst_done = false
5443   local elong_map = Babel.arabic.elong_map
5444   local last_line
5445   local GLYPH = node.id'glyph'
5446   local KASHIDA = luatexbase.registernumber'bblar@kashida'
5447   local LOCALE = luatexbase.registernumber'bbl@attr@locale'
5448
5449   for line in node.traverse_id(node.id'hlist', head) do
5450     % Exclude last line. todo. But-- it discards one-word lines, too!
5451     % ? Look for glue = 12:15
5452     if (line.glue_sign == 1 and line.glue_order == 0) then
5453       elongs = {}     % Stores elongated candidates of each line
5454       k_list = {}     % And all letters with kashida
5455       pos_inline = 0  % Not yet used
5456
5457       for n in node.traverse_id(GLYPH, line.head) do
5458         pos_inline = pos_inline + 1 % To find where it is. Not used.
5459
5460         % Elongated glyphs
5461         if elong_map then
5462           local locale = node.get_attribute(n, LOCALE)
5463           if elong_map[locale] and elong_map[locale][n.font] and
5464               elong_map[locale][n.font][n.char] then
5465             table.insert(elongs, {node = n, locale = locale} )
5466             node.set_attribute(n.prev, KASHIDA, 0)
5467           end
5468         end
5469
5470         % Tatwil
```

181

```
5471        if Babel.kashida_wts then
5472          local k_wt = node.get_attribute(n, KASHIDA)
5473          if k_wt > 0 then % todo. parameter for multi inserts
5474            table.insert(k_list, {node = n, weight = k_wt, pos = pos_inline})
5475          end
5476        end
5477
5478      end % of node.traverse_id
5479
5480      if #elongs == 0 and #k_list == 0 then goto next_line end
5481
5482      full  = line.width
5483      shift = line.shift
5484      goal  = full * Babel.arabic.justify_factor % A bit crude
5485      width = node.dimensions(line.head)     % The 'natural' width
5486
5487      % == Elongated ==
5488      % Original idea taken from 'chikenize'
5489      while (#elongs > 0 and width < goal) do
5490        subst_done = true
5491        local x = #elongs
5492        local curr = elongs[x].node
5493        local oldchar = curr.char
5494        curr.char = elong_map[elongs[x].locale][curr.font][curr.char]
5495        width = node.dimensions(line.head)  % Check if the line is too wide
5496        % Substitute back if the line would be too wide and break:
5497        if width > goal then
5498          curr.char = oldchar
5499          break
5500        end
5501        % If continue, pop the just substituted node from the list:
5502        table.remove(elongs, x)
5503      end
5504
5505      % == Tatwil ==
5506      if #k_list == 0 then goto next_line end
5507
5508      width = node.dimensions(line.head)     % The 'natural' width
5509      k_curr = #k_list
5510      wt_pos = 1
5511
5512      while width < goal do
5513        subst_done = true
5514        k_item = k_list[k_curr].node
5515        if k_list[k_curr].weight == Babel.kashida_wts[wt_pos] then
5516          d = node.copy(k_item)
5517          d.char = 0x0640
5518          line.head, new = node.insert_after(line.head, k_item, d)
5519          width_new = node.dimensions(line.head)
5520          if width > goal or width == width_new then
5521            node.remove(line.head, new) % Better compute before
5522            break
5523          end
5524          width = width_new
5525        end
5526        if k_curr == 1 then
5527          k_curr = #k_list
5528          wt_pos = (wt_pos >= table.getn(Babel.kashida_wts)) and 1 or wt_pos+1
5529        else
```

182

```
5530              k_curr = k_curr - 1
5531          end
5532      end
5533
5534      ::next_line::
5535
5536      % Must take into account marks and ins, see luatex manual.
5537      % Have to be executed only if there are changes. Investigate
5538      % what's going on exactly.
5539      if subst_done then
5540        d = node.hpack(line.head, full, 'exactly')
5541        d.shift = shift
5542        node.insert_before(head, line, d)
5543        node.remove(head, line)
5544      end
5545    end % if process line
5546  end % for lines
5547  return head
5548 end
5549 }
5550 \endgroup
5551 \fi\fi % Arabic just block
```

13.7   Common stuff

```
5552 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
5553 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
5554 \DisableBabelHook{babel-fontspec}
5555 ⟨⟨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.

```
5556 % TODO - to a lua file
5557 \directlua{
5558 Babel.script_blocks = {
5559   ['dflt'] = {},
5560   ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
5561               {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
5562   ['Armn'] = {{0x0530, 0x058F}},
5563   ['Beng'] = {{0x0980, 0x09FF}},
5564   ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
5565   ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
5566   ['Cyrl'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
5567               {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
5568   ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
5569   ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
5570               {0xAB00, 0xAB2F}},
5571   ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
5572   % Don't follow strictly Unicode, which places some Coptic letters in
5573   % the 'Greek and Coptic' block
5574   ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
5575   ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
5576               {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
```

```
5577                {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
5578                {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
5579                {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
5580                {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
5581   ['Hebr'] = {{0x0590, 0x05FF}},
5582   ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
5583                {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
5584   ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
5585   ['Knda'] = {{0x0C80, 0x0CFF}},
5586   ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
5587                {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
5588                {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
5589   ['Laoo'] = {{0x0E80, 0x0EFF}},
5590   ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
5591                {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
5592                {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
5593   ['Mahj'] = {{0x11150, 0x1117F}},
5594   ['Mlym'] = {{0x0D00, 0x0D7F}},
5595   ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
5596   ['Orya'] = {{0x0B00, 0x0B7F}},
5597   ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
5598   ['Syrc'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
5599   ['Taml'] = {{0x0B80, 0x0BFF}},
5600   ['Telu'] = {{0x0C00, 0x0C7F}},
5601   ['Tfng'] = {{0x2D30, 0x2D7F}},
5602   ['Thai'] = {{0x0E00, 0x0E7F}},
5603   ['Tibt'] = {{0x0F00, 0x0FFF}},
5604   ['Vaii'] = {{0xA500, 0xA63F}},
5605   ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
5606 }
5607
5608 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
5609 Babel.script_blocks.Hant = Babel.script_blocks.Hans
5610 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
5611
5612 function Babel.locale_map(head)
5613   if not Babel.locale_mapped then return head end
5614
5615   local LOCALE = luatexbase.registernumber'bbl@attr@locale'
5616   local GLYPH = node.id('glyph')
5617   local inmath = false
5618   local toloc_save
5619   for item in node.traverse(head) do
5620     local toloc
5621     if not inmath and item.id == GLYPH then
5622       % Optimization: build a table with the chars found
5623       if Babel.chr_to_loc[item.char] then
5624         toloc = Babel.chr_to_loc[item.char]
5625       else
5626         for lc, maps in pairs(Babel.loc_to_scr) do
5627           for _, rg in pairs(maps) do
5628             if item.char >= rg[1] and item.char <= rg[2] then
5629               Babel.chr_to_loc[item.char] = lc
5630               toloc = lc
5631               break
5632             end
5633           end
5634         end
5635       end
```

```
5636        % Now, take action, but treat composite chars in a different
5637        % fashion, because they 'inherit' the previous locale. Not yet
5638        % optimized.
5639        if not toloc and
5640            (item.char >= 0x0300 and item.char <= 0x036F) or
5641            (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
5642            (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
5643          toloc = toloc_save
5644        end
5645        if toloc and toloc > -1 then
5646          if Babel.locale_props[toloc].lg then
5647            item.lang = Babel.locale_props[toloc].lg
5648            node.set_attribute(item, LOCALE, toloc)
5649          end
5650          if Babel.locale_props[toloc]['/'..item.font] then
5651            item.font = Babel.locale_props[toloc]['/'..item.font]
5652          end
5653          toloc_save = toloc
5654        end
5655      elseif not inmath and item.id == 7 then
5656        item.replace = item.replace and Babel.locale_map(item.replace)
5657        item.pre     = item.pre and Babel.locale_map(item.pre)
5658        item.post    = item.post and Babel.locale_map(item.post)
5659      elseif item.id == node.id'math' then
5660        inmath = (item.subtype == 0)
5661      end
5662   end
5663   return head
5664 end
5665 }
```

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

```
5666 \newcommand\babelcharproperty[1]{%
5667   \count@=#1\relax
5668   \ifvmode
5669     \expandafter\bbl@chprop
5670   \else
5671     \bbl@error{\string\babelcharproperty\space can be used only in\\%
5672                vertical mode (preamble or between paragraphs)}%
5673              {See the manual for futher info}%
5674   \fi}
5675 \newcommand\bbl@chprop[3][\the\count@]{%
5676   \@tempcnta=#1\relax
5677   \bbl@ifunset{bbl@chprop@#2}%
5678     {\bbl@error{No property named '#2'. Allowed values are\\%
5679                direction (bc), mirror (bmg), and linebreak (lb)}%
5680              {See the manual for futher info}}%
5681     {}%
5682   \loop
5683     \bbl@cs{chprop@#2}{#3}%
5684   \ifnum\count@<\@tempcnta
5685     \advance\count@\@ne
5686   \repeat}
5687 \def\bbl@chprop@direction#1{%
5688   \directlua{
5689     Babel.characters[\the\count@] =  Babel.characters[\the\count@] or {}
5690     Babel.characters[\the\count@]['d'] = '#1'
5691   }}
```

```
5692 \let\bbl@chprop@bc\bbl@chprop@direction
5693 \def\bbl@chprop@mirror#1{%
5694   \directlua{
5695     Babel.characters[\the\count@] =  Babel.characters[\the\count@] or {}
5696     Babel.characters[\the\count@]['m'] = '\number#1'
5697   }}
5698 \let\bbl@chprop@bmg\bbl@chprop@mirror
5699 \def\bbl@chprop@linebreak#1{%
5700   \directlua{
5701     Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
5702     Babel.cjk_characters[\the\count@]['c'] = '#1'
5703   }}
5704 \let\bbl@chprop@lb\bbl@chprop@linebreak
5705 \def\bbl@chprop@locale#1{%
5706   \directlua{
5707     Babel.chr_to_loc = Babel.chr_to_loc or {}
5708     Babel.chr_to_loc[\the\count@] =
5709       \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@@#1}}\space
5710   }}
```

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.

```
5711 \begingroup % TODO - to a lua file
5712 \catcode`\~=12
5713 \catcode`\#=12
5714 \catcode`\%=12
5715 \catcode`\&=14
5716 \directlua{
5717   Babel.linebreaking.replacements = {}
5718   Babel.linebreaking.replacements[0] = {}  &% pre
5719   Babel.linebreaking.replacements[1] = {}  &% post
5720
5721   &% Discretionaries contain strings as nodes
5722   function Babel.str_to_nodes(fn, matches, base)
5723     local n, head, last
5724     if fn == nil then return nil end
5725     for s in string.utfvalues(fn(matches)) do
5726       if base.id == 7 then
5727         base = base.replace
5728       end
5729       n = node.copy(base)
5730       n.char    = s
5731       if not head then
5732         head = n
5733       else
5734         last.next = n
5735       end
5736       last = n
```

```
5737      end
5738    return head
5739  end
5740
5741  Babel.fetch_subtext = {}
5742
5743  Babel.ignore_pre_char = function(node)
5744    return (node.lang == \the\l@nohyphenation)
5745  end
5746
5747  &% Merging both functions doesn't seen feasible, because there are too
5748  &% many differences.
5749  Babel.fetch_subtext[0] = function(head)
5750    local word_string = ''
5751    local word_nodes = {}
5752    local lang
5753    local item = head
5754    local inmath = false
5755
5756    while item do
5757
5758      if item.id == 11 then
5759        inmath = (item.subtype == 0)
5760      end
5761
5762      if inmath then
5763        &% pass
5764
5765      elseif item.id == 29 then
5766        local locale = node.get_attribute(item, Babel.attr_locale)
5767
5768        if lang == locale or lang == nil then
5769          lang = lang or locale
5770          if Babel.ignore_pre_char(item) then
5771            word_string = word_string .. Babel.us_char
5772          else
5773            word_string = word_string .. unicode.utf8.char(item.char)
5774          end
5775          word_nodes[#word_nodes+1] = item
5776        else
5777          break
5778        end
5779
5780      elseif item.id == 12 and item.subtype == 13 then
5781        word_string = word_string .. ' '
5782        word_nodes[#word_nodes+1] = item
5783
5784      &% Ignore leading unrecognized nodes, too.
5785      elseif word_string ~= '' then
5786        word_string = word_string .. Babel.us_char
5787        word_nodes[#word_nodes+1] = item  &% Will be ignored
5788      end
5789
5790      item = item.next
5791    end
5792
5793    &% Here and above we remove some trailing chars but not the
5794    &% corresponding nodes. But they aren't accessed.
5795    if word_string:sub(-1) == ' ' then
```

187

```
5796        word_string = word_string:sub(1,-2)
5797      end
5798      word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
5799      return word_string, word_nodes, item, lang
5800    end
5801
5802    Babel.fetch_subtext[1] = function(head)
5803      local word_string = ''
5804      local word_nodes = {}
5805      local lang
5806      local item = head
5807      local inmath = false
5808
5809      while item do
5810
5811        if item.id == 11 then
5812          inmath = (item.subtype == 0)
5813        end
5814
5815        if inmath then
5816          &% pass
5817
5818        elseif item.id == 29 then
5819          if item.lang == lang or lang == nil then
5820            if (item.char ~= 124) and (item.char ~= 61) then &% not =, not |
5821              lang = lang or item.lang
5822              word_string = word_string .. unicode.utf8.char(item.char)
5823              word_nodes[#word_nodes+1] = item
5824            end
5825          else
5826            break
5827          end
5828
5829        elseif item.id == 7 and item.subtype == 2 then
5830          word_string = word_string .. '='
5831          word_nodes[#word_nodes+1] = item
5832
5833        elseif item.id == 7 and item.subtype == 3 then
5834          word_string = word_string .. '|'
5835          word_nodes[#word_nodes+1] = item
5836
5837        &% (1) Go to next word if nothing was found, and (2) implictly
5838        &% remove leading USs.
5839        elseif word_string == '' then
5840          &% pass
5841
5842        &% This is the responsible for splitting by words.
5843        elseif (item.id == 12 and item.subtype == 13) then
5844          break
5845
5846        else
5847          word_string = word_string .. Babel.us_char
5848          word_nodes[#word_nodes+1] = item  &% Will be ignored
5849        end
5850
5851        item = item.next
5852      end
5853
5854      word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
```

```lua
5855      return word_string, word_nodes, item, lang
5856    end
5857
5858    function Babel.pre_hyphenate_replace(head)
5859      Babel.hyphenate_replace(head, 0)
5860    end
5861
5862    function Babel.post_hyphenate_replace(head)
5863      Babel.hyphenate_replace(head, 1)
5864    end
5865
5866    function Babel.debug_hyph(w, wn, sc, first, last, last_match)
5867      local ss = ''
5868      for pp = 1, 40 do
5869        if wn[pp] then
5870          if wn[pp].id == 29 then
5871            ss = ss .. unicode.utf8.char(wn[pp].char)
5872          else
5873            ss = ss .. '{' .. wn[pp].id .. '}'
5874          end
5875        end
5876      end
5877      print('nod', ss)
5878      print('lst_m',
5879        string.rep(' ', unicode.utf8.len(
5880          string.sub(w, 1, last_match))-1) .. '>')
5881      print('str', w)
5882      print('sc', string.rep(' ', sc-1) .. '^')
5883      if first == last then
5884        print('f=l', string.rep(' ', first-1) .. '!')
5885      else
5886        print('f/l', string.rep(' ', first-1) .. '[' ..
5887          string.rep(' ', last-first-1) .. ']')
5888      end
5889    end
5890
5891    Babel.us_char = string.char(31)
5892
5893    function Babel.hyphenate_replace(head, mode)
5894      local u = unicode.utf8
5895      local lbkr = Babel.linebreaking.replacements[mode]
5896
5897      local word_head = head
5898
5899      while true do  &% for each subtext block
5900
5901        local w, w_nodes, nw, lang = Babel.fetch_subtext[mode](word_head)
5902
5903        if Babel.debug then
5904          print()
5905          print((mode == 0) and '@@@@<' or '@@@@>', w)
5906        end
5907
5908        if nw == nil and w == '' then break end
5909
5910        if not lang then goto next end
5911        if not lbkr[lang] then goto next end
5912
5913        &% For each saved (pre|post)hyphenation. TODO. Reconsider how
```

```
5914        &% loops are nested.
5915        for k=1, #lbkr[lang] do
5916          local p = lbkr[lang][k].pattern
5917          local r = lbkr[lang][k].replace
5918
5919          if Babel.debug then
5920            print('*****', p, mode)
5921          end
5922
5923          &% This variable is set in some cases below to the first *byte*
5924          &% after the match, either as found by u.match (faster) or the
5925          &% computed position based on sc if w has changed.
5926          local last_match = 0
5927          local step = 0
5928
5929          &% For every match.
5930          while true do
5931            if Babel.debug then
5932              print('=====')
5933            end
5934            local new  &% used when inserting and removing nodes
5935
5936            local matches = { u.match(w, p, last_match) }
5937
5938            if #matches < 2 then break end
5939
5940            &% Get and remove empty captures (with ()'s, which return a
5941            &% number with the position), and keep actual captures
5942            &% (from (...)), if any, in matches.
5943            local first = table.remove(matches, 1)
5944            local last  = table.remove(matches, #matches)
5945            &% Non re-fetched substrings may contain \31, which separates
5946            &% subsubstrings.
5947            if string.find(w:sub(first, last-1), Babel.us_char) then break end
5948
5949            local save_last = last &% with A()BC()D, points to D
5950
5951            &% Fix offsets, from bytes to unicode. Explained above.
5952            first = u.len(w:sub(1, first-1)) + 1
5953            last  = u.len(w:sub(1, last-1)) &% now last points to C
5954
5955            &% This loop stores in n small table the nodes
5956            &% corresponding to the pattern. Used by 'data' to provide a
5957            &% predictable behavior with 'insert' (now w_nodes is modified on
5958            &% the fly), and also access to 'remove'd nodes.
5959            local sc = first-1            &% Used below, too
5960            local data_nodes = {}
5961
5962            for q = 1, last-first+1 do
5963              data_nodes[q] = w_nodes[sc+q]
5964            end
5965
5966            &% This loop traverses the matched substring and takes the
5967            &% corresponding action stored in the replacement list.
5968            &% sc = the position in substr nodes / string
5969            &% rc = the replacement table index
5970            local rc = 0
5971
5972            while rc < last-first+1 do &% for each replacement
```

```
5973            if Babel.debug then
5974              print('.....', rc + 1)
5975            end
5976            sc = sc + 1
5977            rc = rc + 1
5978
5979            if Babel.debug then
5980              Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
5981              local ss = ''
5982              for itt in node.traverse(head) do
5983               if itt.id == 29 then
5984                 ss = ss .. unicode.utf8.char(itt.char)
5985               else
5986                 ss = ss .. '{' .. itt.id .. '}'
5987               end
5988              end
5989              print('*****************', ss)
5990
5991            end
5992
5993            local crep = r[rc]
5994            local item = w_nodes[sc]
5995            local item_base = item
5996            local placeholder = Babel.us_char
5997            local d
5998
5999            if crep and crep.data then
6000              item_base = data_nodes[crep.data]
6001            end
6002
6003            if crep then
6004              step = crep.step or 0
6005            end
6006
6007            if crep and next(crep) == nil then &% = {}
6008              last_match = save_last    &% Optimization
6009              goto next
6010
6011            elseif crep == nil or crep.remove then
6012              node.remove(head, item)
6013              table.remove(w_nodes, sc)
6014              w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
6015              sc = sc - 1  &% Nothing has been inserted.
6016              last_match = utf8.offset(w, sc+1+step)
6017              goto next
6018
6019            elseif crep and crep.kashida then &% Experimental
6020              node.set_attribute(item,
6021                luatexbase.registernumber'bblar@kashida',
6022                crep.kashida)
6023              last_match = utf8.offset(w, sc+1+step)
6024              goto next
6025
6026            elseif crep and crep.string then
6027              local str = crep.string(matches)
6028              if str == '' then  &% Gather with nil
6029                node.remove(head, item)
6030                table.remove(w_nodes, sc)
6031                w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
```

```
6032                    sc = sc - 1   &% Nothing has been inserted.
6033                else
6034                  local loop_first = true
6035                  for s in string.utfvalues(str) do
6036                    d = node.copy(item_base)
6037                    d.char = s
6038                    if loop_first then
6039                      loop_first = false
6040                      head, new = node.insert_before(head, item, d)
6041                      if sc == 1 then
6042                        word_head = head
6043                      end
6044                      w_nodes[sc] = d
6045                      w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc+1)
6046                    else
6047                      sc = sc + 1
6048                      head, new = node.insert_before(head, item, d)
6049                      table.insert(w_nodes, sc, new)
6050                      w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc)
6051                    end
6052                    if Babel.debug then
6053                      print('.....', 'str')
6054                      Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6055                    end
6056                  end   &% for
6057                  node.remove(head, item)
6058                end   &% if ''
6059                last_match = utf8.offset(w, sc+1+step)
6060                goto next
6061
6062            elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
6063              d = node.new(7, 0)    &% (disc, discretionary)
6064              d.pre     = Babel.str_to_nodes(crep.pre, matches, item_base)
6065              d.post    = Babel.str_to_nodes(crep.post, matches, item_base)
6066              d.replace = Babel.str_to_nodes(crep.no, matches, item_base)
6067              d.attr = item_base.attr
6068              if crep.pre == nil then   &% TeXbook p96
6069                d.penalty = crep.penalty or tex.hyphenpenalty
6070              else
6071                d.penalty = crep.penalty or tex.exhyphenpenalty
6072              end
6073              placeholder = '|'
6074              head, new = node.insert_before(head, item, d)
6075
6076            elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
6077              &% ERROR
6078
6079            elseif crep and crep.penalty then
6080              d = node.new(14, 0)   &% (penalty, userpenalty)
6081              d.attr = item_base.attr
6082              d.penalty = crep.penalty
6083              head, new = node.insert_before(head, item, d)
6084
6085            elseif crep and crep.space then
6086              &% 655360 = 10 pt = 10 * 65536 sp
6087              d = node.new(12, 13)     &% (glue, spaceskip)
6088              local quad = font.getfont(item_base.font).size or 655360
6089              node.setglue(d, crep.space[1] * quad,
6090                              crep.space[2] * quad,
```

192

```
6091                               crep.space[3] * quad)
6092                if mode == 0 then
6093                  placeholder = ' '
6094                end
6095                head, new = node.insert_before(head, item, d)
6096
6097            elseif crep and crep.spacefactor then
6098                d = node.new(12, 13)      &% (glue, spaceskip)
6099                local base_font = font.getfont(item_base.font)
6100                node.setglue(d,
6101                  crep.spacefactor[1] * base_font.parameters['space'],
6102                  crep.spacefactor[2] * base_font.parameters['space_stretch'],
6103                  crep.spacefactor[3] * base_font.parameters['space_shrink'])
6104                if mode == 0 then
6105                  placeholder = ' '
6106                end
6107                head, new = node.insert_before(head, item, d)
6108
6109            elseif mode == 0 and crep and crep.space then
6110                &% ERROR
6111
6112            end  &% ie replacement cases
6113
6114            &% Shared by disc, space and penalty.
6115            if sc == 1 then
6116              word_head = head
6117            end
6118            if crep.insert then
6119              w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc)
6120              table.insert(w_nodes, sc, new)
6121              last = last + 1
6122            else
6123              w_nodes[sc] = d
6124              node.remove(head, item)
6125              w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1)
6126            end
6127
6128            last_match = utf8.offset(w, sc+1+step)
6129
6130            ::next::
6131
6132          end  &% for each replacement
6133
6134          if Babel.debug then
6135              print('.....', '/')
6136              Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6137          end
6138
6139        end  &% for match
6140
6141      end  &% for patterns
6142
6143    ::next::
6144    word_head = nw
6145  end  &% for substring
6146  return head
6147 end
6148
6149 &% This table stores capture maps, numbered consecutively
```

```
6150   Babel.capture_maps = {}
6151
6152   &% The following functions belong to the next macro
6153   function Babel.capture_func(key, cap)
6154     local ret = "[[" .. cap:gsub('{([0-9])}', "]]..m[%1]..[[") .. "]]"
6155     local cnt
6156     local u = unicode.utf8
6157     ret, cnt = ret:gsub('{([0-9])|([^|]+)|(.-)}', Babel.capture_func_map)
6158     if cnt == 0 then
6159       ret = u.gsub(ret, '{(%x%x%x%x+)}',
6160             function (n)
6161               return u.char(tonumber(n, 16))
6162             end)
6163     end
6164     ret = ret:gsub("%[%[%]%]%.%.", '')
6165     ret = ret:gsub("%.%.%[%[%]%]", '')
6166     return key .. [[=function(m) return ]] .. ret .. [[ end]]
6167   end
6168
6169   function Babel.capt_map(from, mapno)
6170     return Babel.capture_maps[mapno][from] or from
6171   end
6172
6173   &% Handle the {n|abc|ABC} syntax in captures
6174   function Babel.capture_func_map(capno, from, to)
6175     local u = unicode.utf8
6176     from = u.gsub(from, '{(%x%x%x%x+)}',
6177           function (n)
6178             return u.char(tonumber(n, 16))
6179           end)
6180     to = u.gsub(to, '{(%x%x%x%x+)}',
6181           function (n)
6182             return u.char(tonumber(n, 16))
6183           end)
6184     local froms = {}
6185     for s in string.utfcharacters(from) do
6186       table.insert(froms, s)
6187     end
6188     local cnt = 1
6189     table.insert(Babel.capture_maps, {})
6190     local mlen = table.getn(Babel.capture_maps)
6191     for s in string.utfcharacters(to) do
6192       Babel.capture_maps[mlen][froms[cnt]] = s
6193       cnt = cnt + 1
6194     end
6195     return "]]..Babel.capt_map(m[" .. capno .. "]," ..
6196           (mlen) .. ").." .. "[["
6197   end
6198
6199   &% Create/Extend reversed sorted list of kashida weights:
6200   function Babel.capture_kashida(key, wt)
6201     wt = tonumber(wt)
6202     if Babel.kashida_wts then
6203       for p, q in ipairs(Babel.kashida_wts) do
6204         if wt  == q then
6205           break
6206         elseif wt > q then
6207           table.insert(Babel.kashida_wts, p, wt)
6208           break
```

```
6209        elseif table.getn(Babel.kashida_wts) == p then
6210            table.insert(Babel.kashida_wts, wt)
6211          end
6212        end
6213      else
6214        Babel.kashida_wts = { wt }
6215      end
6216      return 'kashida = ' .. wt
6217    end
6218 }
```

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).

```
6219 \catcode`\#=6
6220 \gdef\babelposthyphenation#1#2#3{&%
6221    \bbl@activateposthyphen
6222    \begingroup
6223      \def\babeltempa{\bbl@add@list\babeltempb}&%
6224      \let\babeltempb\@empty
6225      \def\bbl@tempa{#3}&% TODO. Ugly trick to preserve {}:
6226      \bbl@replace\bbl@tempa{,}{ ,}&%
6227      \expandafter\bbl@foreach\expandafter{\bbl@tempa}{&%
6228        \bbl@ifsamestring{##1}{remove}&%
6229          {\bbl@add@list\babeltempb{nil}}&%
6230          {\directlua{
6231              local rep = [=[##1]=]
6232              rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
6233              rep = rep:gsub('^%s*(insert)%s*,', 'insert = true, ')
6234              rep = rep:gsub(    '(no)%s*=%s*([^%s,]*)', Babel.capture_func)
6235              rep = rep:gsub(   '(pre)%s*=%s*([^%s,]*)', Babel.capture_func)
6236              rep = rep:gsub(  '(post)%s*=%s*([^%s,]*)', Babel.capture_func)
6237              rep = rep:gsub('(string)%s*=%s*([^%s,]*)', Babel.capture_func)
6238              tex.print([[\string\babeltempa{{]] .. rep .. [[}}]])
6239          }}}&%
6240    \directlua{
6241      local lbkr = Babel.linebreaking.replacements[1]
6242      local u = unicode.utf8
6243      local id = \the\csname l@#1\endcsname
6244      &% Convert pattern:
6245      local patt = string.gsub([==[#2]==], '%s', '')
6246      if not u.find(patt, '()', nil, true) then
6247        patt = '()' .. patt .. '()'
6248      end
6249      patt = string.gsub(patt, '%(%)%^', '^()')
6250      patt = string.gsub(patt, '%$%(%)', '()$')
6251      patt = u.gsub(patt, '{(.)}',
6252              function (n)
6253                return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
6254              end)
6255      patt = u.gsub(patt, '{(%x%x%x%x+)}',
6256              function (n)
6257                return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%1')
```

```
6258                end)
6259            lbkr[id] = lbkr[id] or {}
6260            table.insert(lbkr[id], { pattern = patt, replace = { \babeltempb } })
6261        }&%
6262    \endgroup}
6263 % TODO. Copypaste pattern.
6264 \gdef\babelprehyphenation#1#2#3{&%
6265    \bbl@activateprehyphen
6266    \begingroup
6267      \def\babeltempa{\bbl@add@list\babeltempb}&%
6268      \let\babeltempb\@empty
6269      \def\bbl@tempa{#3}&% TODO. Ugly trick to preserve {}:
6270      \bbl@replace\bbl@tempa{,}{ ,}&%
6271      \expandafter\bbl@foreach\expandafter{\bbl@tempa}{&%
6272        \bbl@ifsamestring{##1}{remove}&%
6273          {\bbl@add@list\babeltempb{nil}}&%
6274          {\directlua{
6275              local rep = [=[##1]=]
6276              rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
6277              rep = rep:gsub('^%s*(insert)%s*,', 'insert = true, ')
6278              rep = rep:gsub('(string)%s*=%s*([^%s,]*)', Babel.capture_func)
6279              rep = rep:gsub('(space)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
6280                'space = {' .. '%2, %3, %4' .. '}')
6281              rep = rep:gsub('(spacefactor)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
6282                'spacefactor = {' .. '%2, %3, %4' .. '}')
6283              rep = rep:gsub('(kashida)%s*=%s*([^%s,]*)', Babel.capture_kashida)
6284              tex.print([[\string\babeltempa{{]] .. rep .. [[}]]])
6285          }}}&%
6286      \directlua{
6287        local lbkr = Babel.linebreaking.replacements[0]
6288        local u = unicode.utf8
6289        local id = \the\csname bbl@id@@#1\endcsname
6290        &% Convert pattern:
6291        local patt = string.gsub([==[#2]==], '%s', '')
6292        local patt = string.gsub(patt, '|', ' ')
6293        if not u.find(patt, '()', nil, true) then
6294          patt = '()' .. patt .. '()'
6295        end
6296        &% patt = string.gsub(patt, '%(%)%^', '^()')
6297        &% patt = string.gsub(patt, '([^%%])%$%(%)', '%1()$')
6298        patt = u.gsub(patt, '{(.)}',
6299                function (n)
6300                  return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
6301                end)
6302        patt = u.gsub(patt, '{(%x%x%x%x+)}',
6303                function (n)
6304                  return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%%1')
6305                end)
6306        lbkr[id] = lbkr[id] or {}
6307        table.insert(lbkr[id], { pattern = patt, replace = { \babeltempb } })
6308      }&%
6309    \endgroup}
6310 \endgroup
6311 \def\bbl@activateposthyphen{%
6312    \let\bbl@activateposthyphen\relax
6313    \directlua{
6314      Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
6315    }}
6316 \def\bbl@activateprehyphen{%
```

```
6317    \let\bbl@activateprehyphen\relax
6318    \directlua{
6319      Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
6320    }}
```

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.

```
6321 \bbl@trace{Redefinitions for bidi layout}
6322 \ifx\@eqnnum\@undefined\else
6323   \ifx\bbl@attr@dir\@undefined\else
6324     \edef\@eqnnum{{%
6325       \unexpanded{\ifcase\bbl@attr@dir\else\bbl@textdir\@ne\fi}%
6326       \unexpanded\expandafter{\@eqnnum}}}
6327   \fi
6328 \fi
6329 \ifx\bbl@opt@layout\@nnil\endinput\fi  % if no layout
6330 \ifnum\bbl@bidimode>\z@
6331   \def\bbl@nextfake#1{%  non-local changes, use always inside a group!
6332     \bbl@exp{%
6333       \mathdir\the\bodydir
6334       #1%                Once entered in math, set boxes to restore values
6335       \<ifmmode>%
6336         \everyvbox{%
6337           \the\everyvbox
6338           \bodydir\the\bodydir
6339           \mathdir\the\mathdir
6340           \everyhbox{\the\everyhbox}%
6341           \everyvbox{\the\everyvbox}}%
6342         \everyhbox{%
6343           \the\everyhbox
6344           \bodydir\the\bodydir
6345           \mathdir\the\mathdir
6346           \everyhbox{\the\everyhbox}%
6347           \everyvbox{\the\everyvbox}}%
6348       \<fi>}}%
6349   \def\@hangfrom#1{%
6350     \setbox\@tempboxa\hbox{{#1}}%
6351     \hangindent\wd\@tempboxa
6352     \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
6353       \shapemode\@ne
6354     \fi
6355     \noindent\box\@tempboxa}
6356 \fi
6357 \IfBabelLayout{tabular}
6358   {\let\bbl@OL@@tabular\@tabular
6359    \bbl@replace\@tabular{$}{\bbl@nextfake$}%
```

```
6360     \let\bbl@NL@@tabular\@tabular
6361     \AtBeginDocument{%
6362       \ifx\bbl@NL@@tabular\@tabular\else
6363         \bbl@replace\@tabular{$}{\bbl@nextfake$}%
6364         \let\bbl@NL@@tabular\@tabular
6365       \fi}}
6366     {}
6367 \IfBabelLayout{lists}
6368     {\let\bbl@OL@list\list
6369      \bbl@sreplace\list{\parshape}{\bbl@listparshape}%
6370      \let\bbl@NL@list\list
6371      \def\bbl@listparshape#1#2#3{%
6372        \parshape #1 #2 #3 %
6373        \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
6374          \shapemode\tw@
6375        \fi}}
6376     {}
6377 \IfBabelLayout{graphics}
6378     {\let\bbl@pictresetdir\relax
6379      \def\bbl@pictsetdir#1{%
6380        \ifcase\bbl@thetextdir
6381          \let\bbl@pictresetdir\relax
6382        \else
6383          \ifcase#1\bodydir TLT  % Remember this sets the inner boxes
6384            \or\textdir TLT
6385            \else\bodydir TLT \textdir TLT
6386          \fi
6387          % \(text|par)dir required in pgf:
6388          \def\bbl@pictresetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}%
6389        \fi}%
6390      \ifx\AddToHook\@undefined\else
6391        \AddToHook{env/picture/begin}{\bbl@pictsetdir\tw@}%
6392        \directlua{
6393          Babel.get_picture_dir = true
6394          Babel.picture_has_bidi = 0
6395          function Babel.picture_dir (head)
6396            if not Babel.get_picture_dir then return head end
6397            for item in node.traverse(head) do
6398              if item.id == node.id'glyph' then
6399                local itemchar = item.char
6400                % TODO. Copypaste pattern from Babel.bidi (-r)
6401                local chardata = Babel.characters[itemchar]
6402                local dir = chardata and chardata.d or nil
6403                if not dir then
6404                  for nn, et in ipairs(Babel.ranges) do
6405                    if itemchar < et[1] then
6406                      break
6407                    elseif itemchar <= et[2] then
6408                      dir = et[3]
6409                      break
6410                    end
6411                  end
6412                end
6413                if dir and (dir == 'al' or dir == 'r') then
6414                  Babel.picture_has_bidi = 1
6415                end
6416              end
6417            end
6418            return head
```

```
6419        end
6420        luatexbase.add_to_callback("hpack_filter", Babel.picture_dir,
6421          "Babel.picture_dir")
6422      }%
6423    \AtBeginDocument{%
6424      \long\def\put(#1,#2)#3{%
6425        \@killglue
6426        % Try:
6427        \ifx\bbl@pictresetdir\relax
6428          \def\bbl@tempc{0}%
6429        \else
6430          \directlua{
6431            Babel.get_picture_dir = true
6432            Babel.picture_has_bidi = 0
6433          }%
6434          \setbox\z@\hb@xt@\z@{%
6435            \@defaultunitsset\@tempdimc{#1}\unitlength
6436            \kern\@tempdimc
6437            #3\hss}%
6438          \edef\bbl@tempc{\directlua{tex.print(Babel.picture_has_bidi)}}%
6439        \fi
6440        % Do:
6441        \@defaultunitsset\@tempdimc{#2}\unitlength
6442        \raise\@tempdimc\hb@xt@\z@{%
6443          \@defaultunitsset\@tempdimc{#1}\unitlength
6444          \kern\@tempdimc
6445          {\ifnum\bbl@tempc>\z@\bbl@pictresetdir\fi#3}\hss}%
6446        \ignorespaces}%
6447      \MakeRobust\put}%
6448    \fi
6449    \AtBeginDocument
6450      {\ifx\tikz@atbegin@node\@undefined\else
6451        \ifx\AddToHook\@undefined\else % TODO. Still tentative.
6452          \AddToHook{env/pgfpicture/begin}{\bbl@pictsetdir\@ne}%
6453          \bbl@add\pgfinterruptpicture{\bbl@pictresetdir}%
6454        \fi
6455        \let\bbl@OL@pgfpicture\pgfpicture
6456        \bbl@sreplace\pgfpicture{\pgfpicturetrue}%
6457          {\bbl@pictsetdir\z@\pgfpicturetrue}%
6458        \bbl@add\pgfsys@beginpicture{\bbl@pictsetdir\z@}%
6459        \bbl@add\tikz@atbegin@node{\bbl@pictresetdir}%
6460        \bbl@sreplace\tikz{\begingroup}%
6461          {\begingroup\bbl@pictsetdir\tw@}%
6462      \fi
6463      \ifx\AddToHook\@undefined\else
6464        \AddToHook{env/tcolorbox/begin}{\bbl@pictsetdir\@ne}%
6465      \fi
6466      }}
6467    {}
```

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.

```
6468 \IfBabelLayout{counters}%
6469    {\let\bbl@OL@@textsuperscript\@textsuperscript
6470     \bbl@sreplace\@textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
6471     \let\bbl@latinarabic=\@arabic
6472     \let\bbl@OL@@arabic\@arabic
6473     \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
```

```
6474    \@ifpackagewith{babel}{bidi=default}%
6475      {\let\bbl@asciiroman=\@roman
6476       \let\bbl@OL@@roman\@roman
6477       \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciiroman#1}}}%
6478       \let\bbl@asciiRoman=\@Roman
6479       \let\bbl@OL@@roman\@Roman
6480       \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}%
6481       \let\bbl@OL@labelenumii\labelenumii
6482       \def\labelenumii{)\theenumii(}%
6483       \let\bbl@OL@p@enumiii\p@enumiii
6484       \def\p@enumiii{\p@enumii)\theenumii(}}{}}{}
6485 ⟨⟨Footnote changes⟩⟩
6486 \IfBabelLayout{footnotes}%
6487   {\let\bbl@OL@footnote\footnote
6488    \BabelFootnote\footnote\languagename{}{}%
6489    \BabelFootnote\localfootnote\languagename{}{}%
6490    \BabelFootnote\mainfootnote{}{}{}}
6491   {}
```

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.

```
6492 \IfBabelLayout{extras}%
6493   {\let\bbl@OL@underline\underline
6494    \bbl@sreplace\underline{$\@@underline}{\bbl@nextfake$\@@underline}%
6495    \let\bbl@OL@LaTeX2e\LaTeX2e
6496    \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
6497      \if b\expandafter\@car\f@series\@nil\boldmath\fi
6498      \babelsublr{%
6499        \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}}}
6500   {}
6501 ⟨/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.

```
6502 ⟨∗basic-r⟩
6503 Babel = Babel or {}
6504
6505 Babel.bidi_enabled = true
6506
6507 require('babel-data-bidi.lua')
6508
6509 local characters = Babel.characters
6510 local ranges = Babel.ranges
6511
6512 local DIR = node.id("dir")
6513
6514 local function dir_mark(head, from, to, outer)
6515   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
6516   local d = node.new(DIR)
6517   d.dir = '+' .. dir
6518   node.insert_before(head, from, d)
6519   d = node.new(DIR)
6520   d.dir = '-' .. dir
6521   node.insert_after(head, to, d)
6522 end
6523
6524 function Babel.bidi(head, ispar)
6525   local first_n, last_n          -- first and last char with nums
6526   local last_es                  -- an auxiliary 'last' used with nums
6527   local first_d, last_d          -- first and last char in L/R block
6528   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):

```
6529   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
6530   local strong_lr = (strong == 'l') and 'l' or 'r'
6531   local outer = strong
6532
6533   local new_dir = false
6534   local first_dir = false
6535   local inmath = false
6536
6537   local last_lr
6538
6539   local type_n = ''
6540
6541   for item in node.traverse(head) do
6542
6543     -- three cases: glyph, dir, otherwise
6544     if item.id == node.id'glyph'
6545       or (item.id == 7 and item.subtype == 2) then
```

```
6546
6547      local itemchar
6548      if item.id == 7 and item.subtype == 2 then
6549        itemchar = item.replace.char
6550      else
6551        itemchar = item.char
6552      end
6553      local chardata = characters[itemchar]
6554      dir = chardata and chardata.d or nil
6555      if not dir then
6556        for nn, et in ipairs(ranges) do
6557          if itemchar < et[1] then
6558            break
6559          elseif itemchar <= et[2] then
6560            dir = et[3]
6561            break
6562          end
6563        end
6564      end
6565      dir = dir or 'l'
6566      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).

```
6567      if new_dir then
6568        attr_dir = 0
6569        for at in node.traverse(item.attr) do
6570          if at.number == luatexbase.registernumber'bbl@attr@dir' then
6571            attr_dir = at.value % 3
6572          end
6573        end
6574        if attr_dir == 1 then
6575          strong = 'r'
6576        elseif attr_dir == 2 then
6577          strong = 'al'
6578        else
6579          strong = 'l'
6580        end
6581        strong_lr = (strong == 'l') and 'l' or 'r'
6582        outer = strong_lr
6583        new_dir = false
6584      end
6585
6586      if dir == 'nsm' then dir = strong end            -- W1
```

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

```
6587      dir_real = dir              -- We need dir_real to set strong below
6588      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:

```
6589      if strong == 'al' then
6590        if dir == 'en' then dir = 'an' end                -- W2
6591        if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
6592        strong_lr = 'r'                                   -- W3
6593      end
```

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

```
6594    elseif item.id == node.id'dir' and not inmath then
6595      new_dir = true
6596      dir = nil
6597    elseif item.id == node.id'math' then
6598      inmath = (item.subtype == 0)
6599    else
6600      dir = nil            -- Not a char
6601    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>.

```
6602    if dir == 'en' or dir == 'an' or dir == 'et' then
6603      if dir ~= 'et' then
6604        type_n = dir
6605      end
6606      first_n = first_n or item
6607      last_n = last_es or item
6608      last_es = nil
6609    elseif dir == 'es' and last_n then -- W3+W6
6610      last_es = item
6611    elseif dir == 'cs' then                  -- it's right - do nothing
6612    elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
6613      if strong_lr == 'r' and type_n ~= '' then
6614        dir_mark(head, first_n, last_n, 'r')
6615      elseif strong_lr == 'l' and first_d and type_n == 'an' then
6616        dir_mark(head, first_n, last_n, 'r')
6617        dir_mark(head, first_d, last_d, outer)
6618        first_d, last_d = nil, nil
6619      elseif strong_lr == 'l' and type_n ~= '' then
6620        last_d = last_n
6621      end
6622      type_n = ''
6623      first_n, last_n = nil, nil
6624    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:

```
6625    if dir == 'l' or dir == 'r' then
6626      if dir ~= outer then
6627        first_d = first_d or item
6628        last_d = item
6629      elseif first_d and dir ~= strong_lr then
6630        dir_mark(head, first_d, last_d, outer)
6631        first_d, last_d = nil, nil
6632      end
6633    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.

```
6634    if dir and not last_lr and dir ~= 'l' and outer == 'r' then
```

```
6635        item.char = characters[item.char] and
6636                    characters[item.char].m or item.char
6637      elseif (dir or new_dir) and last_lr ~= item then
6638        local mir = outer .. strong_lr .. (dir or outer)
6639        if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
6640          for ch in node.traverse(node.next(last_lr)) do
6641            if ch == item then break end
6642            if ch.id == node.id'glyph' and characters[ch.char] then
6643              ch.char = characters[ch.char].m or ch.char
6644            end
6645          end
6646        end
6647      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`).

```
6648      if dir == 'l' or dir == 'r' then
6649        last_lr = item
6650        strong = dir_real          -- Don't search back - best save now
6651        strong_lr = (strong == 'l') and 'l' or 'r'
6652      elseif new_dir then
6653        last_lr = nil
6654      end
6655    end
```

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

```
6656    if last_lr and outer == 'r' then
6657      for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
6658        if characters[ch.char] then
6659          ch.char = characters[ch.char].m or ch.char
6660        end
6661      end
6662    end
6663    if first_n then
6664      dir_mark(head, first_n, last_n, outer)
6665    end
6666    if first_d then
6667      dir_mark(head, first_d, last_d, outer)
6668    end
```

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

```
6669    return node.prev(head) or head
6670 end
6671 ⟨/basic-r⟩
```

And here the Lua code for bidi=basic:

```
6672 ⟨*basic⟩
6673 Babel = Babel or {}
6674
6675 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
6676
6677 Babel.fontmap = Babel.fontmap or {}
6678 Babel.fontmap[0] = {}       -- l
6679 Babel.fontmap[1] = {}       -- r
6680 Babel.fontmap[2] = {}       -- al/an
6681
6682 Babel.bidi_enabled = true
6683 Babel.mirroring_enabled = true
```

```lua
6684
6685 require('babel-data-bidi.lua')
6686
6687 local characters = Babel.characters
6688 local ranges = Babel.ranges
6689
6690 local DIR = node.id('dir')
6691 local GLYPH = node.id('glyph')
6692
6693 local function insert_implicit(head, state, outer)
6694   local new_state = state
6695   if state.sim and state.eim and state.sim ~= state.eim then
6696     dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
6697     local d = node.new(DIR)
6698     d.dir = '+' .. dir
6699     node.insert_before(head, state.sim, d)
6700     local d = node.new(DIR)
6701     d.dir = '-' .. dir
6702     node.insert_after(head, state.eim, d)
6703   end
6704   new_state.sim, new_state.eim = nil, nil
6705   return head, new_state
6706 end
6707
6708 local function insert_numeric(head, state)
6709   local new
6710   local new_state = state
6711   if state.san and state.ean and state.san ~= state.ean then
6712     local d = node.new(DIR)
6713     d.dir = '+TLT'
6714     _, new = node.insert_before(head, state.san, d)
6715     if state.san == state.sim then state.sim = new end
6716     local d = node.new(DIR)
6717     d.dir = '-TLT'
6718     _, new = node.insert_after(head, state.ean, d)
6719     if state.ean == state.eim then state.eim = new end
6720   end
6721   new_state.san, new_state.ean = nil, nil
6722   return head, new_state
6723 end
6724
6725 -- TODO - \hbox with an explicit dir can lead to wrong results
6726 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
6727 -- was s made to improve the situation, but the problem is the 3-dir
6728 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
6729 -- well.
6730
6731 function Babel.bidi(head, ispar, hdir)
6732   local d    -- d is used mainly for computations in a loop
6733   local prev_d = ''
6734   local new_d = false
6735
6736   local nodes = {}
6737   local outer_first = nil
6738   local inmath = false
6739
6740   local glue_d = nil
6741   local glue_i = nil
6742
```

```
6743    local has_en = false
6744    local first_et = nil
6745
6746    local ATDIR = luatexbase.registernumber'bbl@attr@dir'
6747
6748    local save_outer
6749    local temp = node.get_attribute(head, ATDIR)
6750    if temp then
6751      temp = temp % 3
6752      save_outer = (temp == 0 and 'l') or
6753                   (temp == 1 and 'r') or
6754                   (temp == 2 and 'al')
6755    elseif ispar then           -- Or error? Shouldn't happen
6756      save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
6757    else                        -- Or error? Shouldn't happen
6758      save_outer = ('TRT' == hdir) and 'r' or 'l'
6759    end
6760      -- when the callback is called, we are just _after_ the box,
6761      -- and the textdir is that of the surrounding text
6762    -- if not ispar and hdir ~= tex.textdir then
6763    --   save_outer = ('TRT' == hdir) and 'r' or 'l'
6764    -- end
6765    local outer = save_outer
6766    local last = outer
6767    -- 'al' is only taken into account in the first, current loop
6768    if save_outer == 'al' then save_outer = 'r' end
6769
6770    local fontmap = Babel.fontmap
6771
6772    for item in node.traverse(head) do
6773
6774      -- In what follows, #node is the last (previous) node, because the
6775      -- current one is not added until we start processing the neutrals.
6776
6777      -- three cases: glyph, dir, otherwise
6778      if item.id == GLYPH
6779         or (item.id == 7 and item.subtype == 2) then
6780
6781        local d_font = nil
6782        local item_r
6783        if item.id == 7 and item.subtype == 2 then
6784          item_r = item.replace    -- automatic discs have just 1 glyph
6785        else
6786          item_r = item
6787        end
6788        local chardata = characters[item_r.char]
6789        d = chardata and chardata.d or nil
6790        if not d or d == 'nsm' then
6791          for nn, et in ipairs(ranges) do
6792            if item_r.char < et[1] then
6793              break
6794            elseif item_r.char <= et[2] then
6795              if not d then d = et[3]
6796              elseif d == 'nsm' then d_font = et[3]
6797              end
6798              break
6799            end
6800          end
6801        end
```

```
6802        d = d or 'l'
6803
6804        -- A short 'pause' in bidi for mapfont
6805        d_font = d_font or d
6806        d_font = (d_font == 'l' and 0) or
6807                 (d_font == 'nsm' and 0) or
6808                 (d_font == 'r' and 1) or
6809                 (d_font == 'al' and 2) or
6810                 (d_font == 'an' and 2) or nil
6811      if d_font and fontmap and fontmap[d_font][item_r.font] then
6812        item_r.font = fontmap[d_font][item_r.font]
6813      end
6814
6815      if new_d then
6816        table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
6817        if inmath then
6818          attr_d = 0
6819        else
6820          attr_d = node.get_attribute(item, ATDIR)
6821          attr_d = attr_d % 3
6822        end
6823        if attr_d == 1 then
6824          outer_first = 'r'
6825          last = 'r'
6826        elseif attr_d == 2 then
6827          outer_first = 'r'
6828          last = 'al'
6829        else
6830          outer_first = 'l'
6831          last = 'l'
6832        end
6833        outer = last
6834        has_en = false
6835        first_et = nil
6836        new_d = false
6837      end
6838
6839      if glue_d then
6840        if (d == 'l' and 'l' or 'r') ~= glue_d then
6841            table.insert(nodes, {glue_i, 'on', nil})
6842        end
6843        glue_d = nil
6844        glue_i = nil
6845      end
6846
6847    elseif item.id == DIR then
6848      d = nil
6849      new_d = true
6850
6851    elseif item.id == node.id'glue' and item.subtype == 13 then
6852      glue_d = d
6853      glue_i = item
6854      d = nil
6855
6856    elseif item.id == node.id'math' then
6857      inmath = (item.subtype == 0)
6858
6859    else
6860      d = nil
```

```
6861     end
6862
6863     -- AL <= EN/ET/ES      -- W2 + W3 + W6
6864     if last == 'al' and d == 'en' then
6865       d = 'an'            -- W3
6866     elseif last == 'al' and (d == 'et' or d == 'es') then
6867       d = 'on'           -- W6
6868     end
6869
6870     -- EN + CS/ES + EN       -- W4
6871     if d == 'en' and #nodes >= 2 then
6872       if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
6873           and nodes[#nodes-1][2] == 'en' then
6874         nodes[#nodes][2] = 'en'
6875       end
6876     end
6877
6878     -- AN + CS + AN           -- W4 too, because uax9 mixes both cases
6879     if d == 'an' and #nodes >= 2 then
6880       if (nodes[#nodes][2] == 'cs')
6881           and nodes[#nodes-1][2] == 'an' then
6882         nodes[#nodes][2] = 'an'
6883       end
6884     end
6885
6886     -- ET/EN                  -- W5 + W7->l / W6->on
6887     if d == 'et' then
6888       first_et = first_et or (#nodes + 1)
6889     elseif d == 'en' then
6890       has_en = true
6891       first_et = first_et or (#nodes + 1)
6892     elseif first_et then      -- d may be nil here !
6893       if has_en then
6894         if last == 'l' then
6895           temp = 'l'     -- W7
6896         else
6897           temp = 'en'    -- W5
6898         end
6899       else
6900         temp = 'on'      -- W6
6901       end
6902       for e = first_et, #nodes do
6903         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
6904       end
6905       first_et = nil
6906       has_en = false
6907     end
6908
6909     -- Force mathdir in math if ON (currently works as expected only
6910     -- with 'l')
6911     if inmath and d == 'on' then
6912       d = ('TRT' == tex.mathdir) and 'r' or 'l'
6913     end
6914
6915     if d then
6916       if d == 'al' then
6917         d = 'r'
6918         last = 'al'
6919       elseif d == 'l' or d == 'r' then
```

```
6920        last = d
6921      end
6922      prev_d = d
6923      table.insert(nodes, {item, d, outer_first})
6924    end
6925
6926    outer_first = nil
6927
6928 end
6929
6930 -- TODO -- repeated here in case EN/ET is the last node. Find a
6931 -- better way of doing things:
6932 if first_et then        -- dir may be nil here !
6933   if has_en then
6934     if last == 'l' then
6935       temp = 'l'      -- W7
6936     else
6937       temp = 'en'     -- W5
6938     end
6939   else
6940     temp = 'on'       -- W6
6941   end
6942   for e = first_et, #nodes do
6943     if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
6944   end
6945 end
6946
6947 -- dummy node, to close things
6948 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
6949
6950 --------------  NEUTRAL ----------------
6951
6952 outer = save_outer
6953 last = outer
6954
6955 local first_on = nil
6956
6957 for q = 1, #nodes do
6958   local item
6959
6960   local outer_first = nodes[q][3]
6961   outer = outer_first or outer
6962   last = outer_first or last
6963
6964   local d = nodes[q][2]
6965   if d == 'an' or d == 'en' then d = 'r' end
6966   if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
6967
6968   if d == 'on' then
6969     first_on = first_on or q
6970   elseif first_on then
6971     if last == d then
6972       temp = d
6973     else
6974       temp = outer
6975     end
6976     for r = first_on, q - 1 do
6977       nodes[r][2] = temp
6978       item = nodes[r][1]     -- MIRRORING
```

209

```
6979          if Babel.mirroring_enabled and item.id == GLYPH
6980               and temp == 'r' and characters[item.char] then
6981            local font_mode = font.fonts[item.font].properties.mode
6982            if font_mode ~= 'harf' and font_mode ~= 'plug' then
6983              item.char = characters[item.char].m or item.char
6984            end
6985          end
6986        end
6987        first_on = nil
6988      end
6989
6990      if d == 'r' or d == 'l' then last = d end
6991    end
6992
6993    --------------  IMPLICIT, REORDER ----------------
6994
6995    outer = save_outer
6996    last = outer
6997
6998    local state = {}
6999    state.has_r = false
7000
7001    for q = 1, #nodes do
7002
7003      local item = nodes[q][1]
7004
7005      outer = nodes[q][3] or outer
7006
7007      local d = nodes[q][2]
7008
7009      if d == 'nsm' then d = last end                -- W1
7010      if d == 'en' then d = 'an' end
7011      local isdir = (d == 'r' or d == 'l')
7012
7013      if outer == 'l' and d == 'an' then
7014        state.san = state.san or item
7015        state.ean = item
7016      elseif state.san then
7017        head, state = insert_numeric(head, state)
7018      end
7019
7020      if outer == 'l' then
7021        if d == 'an' or d == 'r' then      -- im -> implicit
7022          if d == 'r' then state.has_r = true end
7023          state.sim = state.sim or item
7024          state.eim = item
7025        elseif d == 'l' and state.sim and state.has_r then
7026          head, state = insert_implicit(head, state, outer)
7027        elseif d == 'l' then
7028          state.sim, state.eim, state.has_r = nil, nil, false
7029        end
7030      else
7031        if d == 'an' or d == 'l' then
7032          if nodes[q][3] then -- nil except after an explicit dir
7033            state.sim = item  -- so we move sim 'inside' the group
7034          else
7035            state.sim = state.sim or item
7036          end
7037          state.eim = item
```

```
7038      elseif d == 'r' and state.sim then
7039        head, state = insert_implicit(head, state, outer)
7040      elseif d == 'r' then
7041        state.sim, state.eim = nil, nil
7042      end
7043    end
7044
7045    if isdir then
7046      last = d           -- Don't search back - best save now
7047    elseif d == 'on' and state.san  then
7048      state.san = state.san or item
7049      state.ean = item
7050    end
7051
7052  end
7053
7054  return node.prev(head) or head
7055 end
7056 ⟨/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.

```
7057 ⟨*nil⟩
7058 \ProvidesLanguage{nil}[⟨⟨date⟩⟩ ⟨⟨version⟩⟩ Nil language]
7059 \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.

```
7060 \ifx\l@nil\@undefined
7061   \newlanguage\l@nil
7062   \@namedef{bbl@hyphendata@\the\l@nil}{{}{}}% Remove warning
7063   \let\bbl@elt\relax
7064   \edef\bbl@languages{%  Add it to the list of languages
7065     \bbl@languages\bbl@elt{nil}{\the\l@nil}{}{}}
7066 \fi
```

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

```
7067 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}
```

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

<div style="margin-left:2em">\captionnil</div>
<div style="margin-left:2em">\datenil</div>

```
7068 \let\captionsnil\@empty
7069 \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.

```
7070 \ldf@finish{nil}
7071 ⟨/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`.

```
7072 ⟨*bplain | blplain⟩
7073 \catcode`\{=1 % left brace is begin-group character
7074 \catcode`\}=2 % right brace is end-group character
7075 \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).

```
7076 \openin 0 hyphen.cfg
7077 \ifeof0
7078 \else
7079   \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.

```
7080   \def\input #1 {%
7081     \let\input\a
7082     \a hyphen.cfg
7083     \let\a\undefined
7084   }
7085 \fi
7086 ⟨/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`.

```
7087 ⟨bplain⟩\a plain.tex
7088 ⟨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.

```
7089 ⟨bplain⟩\def\fmtname{babel-plain}
7090 ⟨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 2ε that are needed for babel.

```
7091 ⟨⟨∗Emulate LaTeX⟩⟩ ≡
7092   % == Code for plain ==
7093 \def\@empty{}
7094 \def\loadlocalcfg#1{%
7095   \openin0#1.cfg
7096   \ifeof0
7097     \closein0
7098   \else
7099     \closein0
7100   {\immediate\write16{***********************************}%
7101    \immediate\write16{* Local config file #1.cfg used}%
7102    \immediate\write16{*}%
7103    }
7104   \input #1.cfg\relax
7105   \fi
7106   \@endofldf}
```

16.3   General tools

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

```
7107 \long\def\@firstofone#1{#1}
7108 \long\def\@firstoftwo#1#2{#1}
7109 \long\def\@secondoftwo#1#2{#2}
7110 \def\@nnil{\@nil}
7111 \def\@gobbletwo#1#2{}
7112 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
7113 \def\@star@or@long#1{%
7114   \@ifstar
7115   {\let\l@ngrel@x\relax#1}%
7116   {\let\l@ngrel@x\long#1}}
7117 \let\l@ngrel@x\relax
7118 \def\@car#1#2\@nil{#1}
7119 \def\@cdr#1#2\@nil{#2}
7120 \let\@typeset@protect\relax
7121 \let\protected@edef\edef
7122 \long\def\@gobble#1{}
7123 \edef\@backslashchar{\expandafter\@gobble\string\\}
7124 \def\strip@prefix#1>{}
7125 \def\g@addto@macro#1#2{{%
7126    \toks@\expandafter{#1#2}%
7127    \xdef#1{\the\toks@}}}
7128 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
7129 \def\@nameuse#1{\csname #1\endcsname}
7130 \def\@ifundefined#1{%
7131   \expandafter\ifx\csname#1\endcsname\relax
7132     \expandafter\@firstoftwo
7133   \else
```

```
7134      \expandafter\@secondoftwo
7135    \fi}
7136 \def\@expandtwoargs#1#2#3{%
7137    \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
7138 \def\zap@space#1 #2{%
7139    #1%
7140    \ifx#2\@empty\else\expandafter\zap@space\fi
7141    #2}
7142 \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}.

```
7143 \ifx\@preamblecmds\@undefined
7144    \def\@preamblecmds{}
7145 \fi
7146 \def\@onlypreamble#1{%
7147    \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
7148      \@preamblecmds\do#1}}
7149 \@onlypreamble\@onlypreamble
```

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

```
7150 \def\begindocument{%
7151    \@begindocumenthook
7152    \global\let\@begindocumenthook\@undefined
7153    \def\do##1{\global\let##1\@undefined}%
7154    \@preamblecmds
7155    \global\let\do\noexpand}
7156 \ifx\@begindocumenthook\@undefined
7157    \def\@begindocumenthook{}
7158 \fi
7159 \@onlypreamble\@begindocumenthook
7160 \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.

```
7161 \def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
7162 \@onlypreamble\AtEndOfPackage
7163 \def\@endofldf{}
7164 \@onlypreamble\@endofldf
7165 \let\bbl@afterlang\@empty
7166 \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.

```
7167 \catcode`\&=\z@
7168 \ifx&if@filesw\@undefined
7169    \expandafter\let\csname if@filesw\expandafter\endcsname
7170      \csname iffalse\endcsname
7171 \fi
7172 \catcode`\&=4
```

Mimick LATEX's commands to define control sequences.

```
7173 \def\newcommand{\@star@or@long\new@command}
7174 \def\new@command#1{%
7175    \@testopt{\@newcommand#1}0}
7176 \def\@newcommand#1[#2]{%
7177    \@ifnextchar [{\@xargdef#1[#2]}%
7178                 {\@argdef#1[#2]}}
```

214

```
7179 \long\def\@argdef#1[#2]#3{%
7180    \@yargdef#1\@ne{#2}{#3}}
7181 \long\def\@xargdef#1[#2][#3]#4{%
7182    \expandafter\def\expandafter#1\expandafter{%
7183       \expandafter\@protected@testopt\expandafter #1%
7184       \csname\string#1\expandafter\endcsname{#3}}%
7185    \expandafter\@yargdef \csname\string#1\endcsname
7186    \tw@{#2}{#4}}
7187 \long\def\@yargdef#1#2#3{%
7188    \@tempcnta#3\relax
7189    \advance \@tempcnta \@ne
7190    \let\@hash@\relax
7191    \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
7192    \@tempcntb #2%
7193    \@whilenum\@tempcntb <\@tempcnta
7194    \do{%
7195       \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
7196       \advance\@tempcntb \@ne}%
7197    \let\@hash@##%
7198    \l@ngrel@x\expandafter\def\expandafter#1\reserved@a}
7199 \def\providecommand{\@star@or@long\provide@command}
7200 \def\provide@command#1{%
7201    \begingroup
7202       \escapechar\m@ne\xdef\@gtempa{{\string#1}}%
7203    \endgroup
7204    \expandafter\@ifundefined\@gtempa
7205       {\def\reserved@a{\new@command#1}}%
7206       {\let\reserved@a\relax
7207        \def\reserved@a{\new@command\reserved@a}}%
7208    \reserved@a}%

7209 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
7210 \def\declare@robustcommand#1{%
7211    \edef\reserved@a{\string#1}%
7212    \def\reserved@b{#1}%
7213    \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
7214    \edef#1{%
7215       \ifx\reserved@a\reserved@b
7216          \noexpand\x@protect
7217          \noexpand#1%
7218       \fi
7219       \noexpand\protect
7220       \expandafter\noexpand\csname
7221          \expandafter\@gobble\string#1 \endcsname
7222    }%
7223    \expandafter\new@command\csname
7224       \expandafter\@gobble\string#1 \endcsname
7225 }
7226 \def\x@protect#1{%
7227    \ifx\protect\@typeset@protect\else
7228       \@x@protect#1%
7229    \fi
7230 }
7231 \catcode`\&=\z@  % Trick to hide conditionals
7232    \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.

```
7233    \def\bbl@tempa{\csname newif\endcsname&ifin@}
```

```
7234 \catcode`\&=4
7235 \ifx\in@\@undefined
7236   \def\in@#1#2{%
7237     \def\in@@##1#1##2##3\in@@{%
7238       \ifx\in@##2\in@false\else\in@true\fi}%
7239     \in@@#2#1\in@\in@@}
7240 \else
7241   \let\bbl@tempa\@empty
7242 \fi
7243 \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).

```
7244 \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.

```
7245 \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.

```
7246 \ifx\@tempcnta\@undefined
7247   \csname newcount\endcsname\@tempcnta\relax
7248 \fi
7249 \ifx\@tempcntb\@undefined
7250   \csname newcount\endcsname\@tempcntb\relax
7251 \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).

```
7252 \ifx\bye\@undefined
7253   \advance\count10 by -2\relax
7254 \fi
7255 \ifx\@ifnextchar\@undefined
7256   \def\@ifnextchar#1#2#3{%
7257     \let\reserved@d=#1%
7258     \def\reserved@a{#2}\def\reserved@b{#3}%
7259     \futurelet\@let@token\@ifnch}
7260   \def\@ifnch{%
7261     \ifx\@let@token\@sptoken
7262       \let\reserved@c\@xifnch
7263     \else
7264       \ifx\@let@token\reserved@d
7265         \let\reserved@c\reserved@a
7266       \else
7267         \let\reserved@c\reserved@b
7268       \fi
7269     \fi
7270     \reserved@c}
7271   \def\:{\let\@sptoken= } \:  % this makes \@sptoken a space token
7272   \def\:{\@xifnch} \expandafter\def\: {\futurelet\@let@token\@ifnch}
7273 \fi
7274 \def\@testopt#1#2{%
7275   \@ifnextchar[{#1}{#1[#2]}}
7276 \def\@protected@testopt#1{%
```

```
7277    \ifx\protect\@typeset@protect
7278      \expandafter\@testopt
7279    \else
7280      \@x@protect#1%
7281    \fi}
7282 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
7283      #2\relax}\fi}
7284 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
7285          \else\expandafter\@gobble\fi{#1}}
```

16.4   Encoding related macros

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

```
7286 \def\DeclareTextCommand{%
7287    \@dec@text@cmd\providecommand
7288 }
7289 \def\ProvideTextCommand{%
7290    \@dec@text@cmd\providecommand
7291 }
7292 \def\DeclareTextSymbol#1#2#3{%
7293    \@dec@text@cmd\chardef#1{#2}#3\relax
7294 }
7295 \def\@dec@text@cmd#1#2#3{%
7296    \expandafter\def\expandafter#2%
7297       \expandafter{%
7298          \csname#3-cmd\expandafter\endcsname
7299          \expandafter#2%
7300          \csname#3\string#2\endcsname
7301       }%
7302 %    \let\@ifdefinable\@rc@ifdefinable
7303    \expandafter#1\csname#3\string#2\endcsname
7304 }
7305 \def\@current@cmd#1{%
7306    \ifx\protect\@typeset@protect\else
7307       \noexpand#1\expandafter\@gobble
7308    \fi
7309 }
7310 \def\@changed@cmd#1#2{%
7311    \ifx\protect\@typeset@protect
7312       \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
7313          \expandafter\ifx\csname ?\string#1\endcsname\relax
7314             \expandafter\def\csname ?\string#1\endcsname{%
7315                \@changed@x@err{#1}%
7316             }%
7317          \fi
7318          \global\expandafter\let
7319             \csname\cf@encoding \string#1\expandafter\endcsname
7320             \csname ?\string#1\endcsname
7321       \fi
7322       \csname\cf@encoding\string#1%
7323          \expandafter\endcsname
7324    \else
7325       \noexpand#1%
7326    \fi
7327 }
7328 \def\@changed@x@err#1{%
7329    \errhelp{Your command will be ignored, type <return> to proceed}%
7330    \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
```

```
7331 \def\DeclareTextCommandDefault#1{%
7332     \DeclareTextCommand#1?%
7333 }
7334 \def\ProvideTextCommandDefault#1{%
7335     \ProvideTextCommand#1?%
7336 }
7337 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
7338 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
7339 \def\DeclareTextAccent#1#2#3{%
7340   \DeclareTextCommand#1{#2}[1]{\accent#3 ##1}
7341 }
7342 \def\DeclareTextCompositeCommand#1#2#3#4{%
7343     \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
7344     \edef\reserved@b{\string##1}%
7345     \edef\reserved@c{%
7346       \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
7347     \ifx\reserved@b\reserved@c
7348       \expandafter\expandafter\expandafter\ifx
7349         \expandafter\@car\reserved@a\relax\relax\@nil
7350         \@text@composite
7351       \else
7352         \edef\reserved@b##1{%
7353           \def\expandafter\noexpand
7354             \csname#2\string#1\endcsname####1{%
7355             \noexpand\@text@composite
7356               \expandafter\noexpand\csname#2\string#1\endcsname
7357               ####1\noexpand\@empty\noexpand\@text@composite
7358               {##1}%
7359           }%
7360         }%
7361         \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
7362       \fi
7363       \expandafter\def\csname\expandafter\string\csname
7364         #2\endcsname\string#1-\string#3\endcsname{#4}
7365     \else
7366       \errhelp{Your command will be ignored, type <return> to proceed}%
7367       \errmessage{\string\DeclareTextCompositeCommand\space used on
7368           inappropriate command \protect#1}
7369     \fi
7370 }
7371 \def\@text@composite#1#2#3\@text@composite{%
7372     \expandafter\@text@composite@x
7373       \csname\string#1-\string#2\endcsname
7374 }
7375 \def\@text@composite@x#1#2{%
7376     \ifx#1\relax
7377         #2%
7378     \else
7379         #1%
7380     \fi
7381 }
7382 %
7383 \def\@strip@args#1:#2-#3\@strip@args{#2}
7384 \def\DeclareTextComposite#1#2#3#4{%
7385     \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
7386     \bgroup
7387         \lccode`\@=#4%
7388         \lowercase{%
7389     \egroup
```

```
7390        \reserved@a @%
7391    }%
7392 }
7393 %
7394 \def\UseTextSymbol#1#2{#2}
7395 \def\UseTextAccent#1#2#3{}
7396 \def\@use@text@encoding#1{}
7397 \def\DeclareTextSymbolDefault#1#2{%
7398     \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
7399 }
7400 \def\DeclareTextAccentDefault#1#2{%
7401     \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
7402 }
7403 \def\cf@encoding{OT1}
```

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

```
7404 \DeclareTextAccent{\"}{OT1}{127}
7405 \DeclareTextAccent{\'}{OT1}{19}
7406 \DeclareTextAccent{\^}{OT1}{94}
7407 \DeclareTextAccent{\`}{OT1}{18}
7408 \DeclareTextAccent{\~}{OT1}{126}
```

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

```
7409 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
7410 \DeclareTextSymbol{\textquotedblright}{OT1}{`\"}
7411 \DeclareTextSymbol{\textquoteleft}{OT1}{`\`}
7412 \DeclareTextSymbol{\textquoteright}{OT1}{`\'}
7413 \DeclareTextSymbol{\i}{OT1}{16}
7414 \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`.

```
7415 \ifx\scriptsize\@undefined
7416   \let\scriptsize\sevenrm
7417 \fi
7418   % End of code for plain
7419 ⟨⟨/Emulate LaTeX⟩⟩
```

A proxy file:

```
7420 ⟨*plain⟩
7421 \input babel.def
7422 ⟨/plain⟩
```

17   Acknowledgements

I would like to thank all who volunteered as β-testers for their time. Michel Goossens supplied contributions for most of the other languages. Nico Poppelier helped polish the text of the documentation and supplied parts of the macros for the Dutch language. Paul Wackers and Werenfried Spit helped find and repair bugs.
During the further development of the babel system I received much help from Bernd Raichle, for which I am grateful.

References

[1] Huda Smitshuijzen Abifares, *Arabic Typography*, Saqi, 2001.

[2] Johannes Braams, Victor Eijkhout and Nico Poppelier, *The development of national LaTeX styles*, *TUGboat* 10 (1989) #3, p. 401–406.

[3] Yannis Haralambous, *Fonts & Encodings*, O'Reilly, 2007.

[4] Donald E. Knuth, *The TEXbook*, Addison-Wesley, 1986.

[5] Jukka K. Korpela, *Unicode Explained*, O'Reilly, 2006.

[6] Leslie Lamport, *LATEX, A document preparation System*, Addison-Wesley, 1986.

[7] Leslie Lamport, in: TEXhax Digest, Volume 89, #13, 17 February 1989.

[8] Ken Lunde, *CJKV Information Processing*, O'Reilly, 2nd ed., 2009.

[9] Hubert Partl, *German TEX*, *TUGboat* 9 (1988) #1, p. 70–72.

[10] Joachim Schrod, *International LATEX is ready to use*, *TUGboat* 11 (1990) #1, p. 87–90.

[11] Apostolos Syropoulos, Antonis Tsolomitis and Nick Sofroniu, *Digital typography using LATEX*, Springer, 2002, p. 301–373.

[12] K.F. Treebus. *Tekstwijzer, een gids voor het grafisch verwerken van tekst*, SDU Uitgeverij ('s-Gravenhage, 1988).