Babel

Localization and internationalization

Unicode
TeX
pdfTeX
LuaTeX
XeTeX

Contents

Troubleshoooting

Part I
User guide

What is this document about? This user guide focuses on internationalization and localization with LaTeX and pdftex, xetex and luatex with the babel package. There are also some notes on its use with 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 wiki. 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.

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 (however, the package inputenc may be omitted with LaTeX $\geq$ 2018-04-01 if the encoding is UTF-8):

```
\documentclass{article}

\usepackage[T1]{fontenc}
% \usepackage[utf8]{inputenc} % Uncomment if LaTeX < 2018-04-01

\usepackage[french]{babel}

\begin{document}

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

\end{document}
```

Now consider something like:

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

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

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

```
\documentclass[russian]{article}

\usepackage{babel}

\babelfont{rm}{DejaVu Serif}

\begin{document}

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

\end{document}
```

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

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

Or the more explanatory:

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

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

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

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

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

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

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

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

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

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 follows. The main language is french, which is activated when the document begins. The package inputenc may be omitted with LaTeX ≥ 2018-04-01 if the encoding is UTF-8.

PDFTEX

```
\documentclass{article}

\usepackage[T1]{fontenc}
\usepackage[utf8]{inputenc}

\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}
```

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 is:

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

\babelfont[russian]{rm}{FreeSerif}

\begin{document}

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

\end{document}
```

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]No predefined "axis" for modifiers are provided because languages and their scripts have quite different needs.

1.5 Troubleshooting

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

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

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

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

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

1.6 Plain

In 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 Plain.[4]

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:

---

[2]In old versions the error read "You have used an old interface to call babel", not very helpful.
[3]In old versions the error read "You haven't loaded the language LANG yet".
[4]Even in the babel kernel there were some macros not compatible with plain. Hopefully these issues have been fixed.

```
\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.

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

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

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

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

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

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

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

1.8   Auxiliary language selectors

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

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

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

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

Spaces after the environment are ignored.

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

Same as \foreignlanguage but as environment. Spaces after the environment are *not* ignored.

This environment was originally intended for intermixing left-to-right typesetting with right-to-left typesetting in engines not supporting a change in the writing direction inside a 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.

\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 discouraged and otherlanguage* (the starred version) is preferred, as the former does not take into account possible changes in encodings of characters like, say, ' done by some languages (eg, italian, french, ukraineb).

To set hyphenation exceptions, use \babelhyphenation (see below).

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.

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 fontenc.[5] 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 three levels of shorthands: *user*, *language*, and *system* (by order of precedence).
Version 3.9 introduces the *language user* level on top of the user level, as described below.
In most cases, you will use only shorthands provided by languages.

NOTE Note the following:

---

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

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) 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 it is 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:

```
\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).[6] 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}}
```

14

| `\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:[7]

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

In addition, the babel core declares ~ as a one-char shorthand which is let, like the standard ~, to a non breaking space.[8]

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

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

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

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

---

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

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

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

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

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

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

1.11 Package options

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

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

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

activegrave Same for `.

shorthands= ⟨char⟩⟨char⟩... | off

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

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

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

safe= none | ref | bib

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

16

`math=`	`active` | `normal`

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

`config=`	⟨*file*⟩

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

`main=`	⟨*language*⟩

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

`headfoot=`	⟨*language*⟩

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

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

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

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

`silent` New 3.9l No warnings and no *infos* are written to the log file.[9]

`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.[10] 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;[11]

---

[9]You can use alternatively the package silence.
[10]Turned off in plain.
[11]Duplicated options count as several ones.

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.[12]

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. 1.23.

layout=

New 3.16 Selects which layout elements are adapted in bidi documents. See sec. 1.23.

1.12   The base **option**

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

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

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

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

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

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

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

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

1.13   ini **files**

An alternative approach to define a language (or, more precisely, a *locale*) is by means of an ini file. Currently babel provides about 200 of these files containing the basic data required for a locale.

---

[12]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.

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 Language 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.

```
\documentclass{book}

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

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

\begin{document}

\tableofcontents

\chapter{სამზარეულო და სუფრის ტრადიციები}

ქართული ტრადიციული სამზარეულო ერთ-ერთი უმდიდრესია მთელ მსოფლიოში.

\end{document}
```

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

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

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

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

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

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

Or also:

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

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

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

Arabic Monolingual documents mostly work in luatex, but it must be fine tuned, and a recent version of fontspec/loaotfload is required. In xetex babel resorts to the bidi package, which seems to work.

Hebrew Niqqud marks seem to work in both engines, but cantillation marks are 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 fine tuning the font behavior is not always possible.

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

```
\babelprovide[import,hyphenrules=+]{lao}
\babelpatterns[lao]{1ກ 1ຍ 1ຶ 1ງ 1ໝ 1ຯ} % 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]	dsb	Lower Sorbian[ul]
agq	Aghem	dua	Duala
ak	Akan	dyo	Jola-Fonyi
am	Amharic[ul]	dz	Dzongkha
ar	Arabic[ul]	ebu	Embu
ar-DZ	Arabic[ul]	ee	Ewe
ar-MA	Arabic[ul]	el	Greek[ul]
ar-SY	Arabic[ul]	el-polyton	Polytonic Greek[ul]
as	Assamese	en-AU	English[ul]
asa	Asu	en-CA	English[ul]
ast	Asturian[ul]	en-GB	English[ul]
az-Cyrl	Azerbaijani	en-NZ	English[ul]
az-Latn	Azerbaijani	en-US	English[ul]
az	Azerbaijani[ul]	en	English[ul]
bas	Basaa	eo	Esperanto[ul]
be	Belarusian[ul]	es-MX	Spanish[ul]
bem	Bemba	es	Spanish[ul]
bez	Bena	et	Estonian[ul]
bg	Bulgarian[ul]	eu	Basque[ul]
bm	Bambara	ewo	Ewondo
bn	Bangla[ul]	fa	Persian[ul]
bo	Tibetan[u]	ff	Fulah
brx	Bodo	fi	Finnish[ul]
bs-Cyrl	Bosnian	fil	Filipino
bs-Latn	Bosnian[ul]	fo	Faroese
bs	Bosnian[ul]	fr	French[ul]
ca	Catalan[ul]	fr-BE	French[ul]
ce	Chechen	fr-CA	French[ul]
cgg	Chiga	fr-CH	French[ul]
chr	Cherokee	fr-LU	French[ul]
ckb	Central Kurdish	fur	Friulian[ul]
cop	Coptic	fy	Western Frisian
cs	Czech[ul]	ga	Irish[ul]
cu	Church Slavic	gd	Scottish Gaelic[ul]
cu-Cyrs	Church Slavic	gl	Galician[ul]
cu-Glag	Church Slavic	grc	Ancient Greek[ul]
cy	Welsh[ul]	gsw	Swiss German
da	Danish[ul]	gu	Gujarati
dav	Taita	guz	Gusii
de-AT	German[ul]	gv	Manx
de-CH	German[ul]	ha-GH	Hausa
de	German[ul]	ha-NE	Hausa[l]
dje	Zarma	ha	Hausa

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
ii	Sichuan Yi	my	Burmese
is	Icelandic[ul]	mzn	Mazanderani
it	Italian[ul]	naq	Nama
ja	Japanese	nb	Norwegian Bokmål[ul]
jgo	Ngomba	nd	North Ndebele
jmc	Machame	ne	Nepali
ka	Georgian[ul]	nl	Dutch[ul]
kab	Kabyle	nmg	Kwasio
kam	Kamba	nn	Norwegian Nynorsk[ul]
kde	Makonde	nnh	Ngiemboon
kea	Kabuverdianu	nus	Nuer
khq	Koyra Chiini	nyn	Nyankole
ki	Kikuyu	om	Oromo
kk	Kazakh	or	Odia
kkj	Kako	os	Ossetic
kl	Kalaallisut	pa-Arab	Punjabi
kln	Kalenjin	pa-Guru	Punjabi
km	Khmer	pa	Punjabi
kn	Kannada[ul]	pl	Polish[ul]
ko	Korean	pms	Piedmontese[ul]
kok	Konkani	ps	Pashto
ks	Kashmiri	pt-BR	Portuguese[ul]
ksb	Shambala	pt-PT	Portuguese[ul]
ksf	Bafia	pt	Portuguese[ul]
ksh	Colognian	qu	Quechua
kw	Cornish	rm	Romansh[ul]
ky	Kyrgyz	rn	Rundi
lag	Langi	ro	Romanian[ul]
lb	Luxembourgish	rof	Rombo
lg	Ganda	ru	Russian[ul]
lkt	Lakota	rw	Kinyarwanda
ln	Lingala	rwk	Rwa
lo	Lao[ul]	sa-Beng	Sanskrit
lrc	Northern Luri	sa-Deva	Sanskrit
lt	Lithuanian[ul]	sa-Gujr	Sanskrit
lu	Luba-Katanga	sa-Knda	Sanskrit
luo	Luo	sa-Mlym	Sanskrit
luy	Luyia	sa-Telu	Sanskrit
lv	Latvian[ul]	sa	Sanskrit
mas	Masai	sah	Sakha
mer	Meru	saq	Samburu
mfe	Morisyen	sbp	Sangu
mg	Malagasy	se	Northern Sami[ul]
mgh	Makhuwa-Meetto	seh	Sena

ses	Koyraboro Senni	twq	Tasawaq
sg	Sango	tzm	Central Atlas Tamazight
shi-Latn	Tachelhit	ug	Uyghur
shi-Tfng	Tachelhit	uk	Ukrainian[ul]
shi	Tachelhit	ur	Urdu[ul]
si	Sinhala	uz-Arab	Uzbek
sk	Slovak[ul]	uz-Cyrl	Uzbek
sl	Slovenian[ul]	uz-Latn	Uzbek
smn	Inari Sami	uz	Uzbek
sn	Shona	vai-Latn	Vai
so	Somali	vai-Vaii	Vai
sq	Albanian[ul]	vai	Vai
sr-Cyrl-BA	Serbian[ul]	vi	Vietnamese[ul]
sr-Cyrl-ME	Serbian[ul]	vun	Vunjo
sr-Cyrl-XK	Serbian[ul]	wae	Walser
sr-Cyrl	Serbian[ul]	xog	Soga
sr-Latn-BA	Serbian[ul]	yav	Yangben
sr-Latn-ME	Serbian[ul]	yi	Yiddish
sr-Latn-XK	Serbian[ul]	yo	Yoruba
sr-Latn	Serbian[ul]	yue	Cantonese
sr	Serbian[ul]	zgh	Standard Moroccan Tamazight
sv	Swedish[ul]		
sw	Swahili	zh-Hans-HK	Chinese
ta	Tamil[u]	zh-Hans-MO	Chinese
te	Telugu[ul]	zh-Hans-SG	Chinese
teo	Teso	zh-Hans	Chinese
th	Thai[ul]	zh-Hant-HK	Chinese
ti	Tigrinya	zh-Hant-MO	Chinese
tk	Turkmen[ul]	zh-Hant	Chinese
to	Tongan	zh	Chinese
tr	Turkish[ul]	zu	Zulu

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

aghem	assamese
akan	asturian
albanian	asu
american	australian
amharic	austrian
ancientgreek	azerbaijani-cyrillic
arabic	azerbaijani-cyrl
arabic-algeria	azerbaijani-latin
arabic-DZ	azerbaijani-latn
arabic-morocco	azerbaijani
arabic-MA	bafia
arabic-syria	bambara
arabic-SY	basaa
armenian	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[13]

churchsslavic-glag

churchsslavic-glagolitic

colognian

cornish

croatian

czech

danish

duala

dutch

dzongkha

embu

english-au

english-australia

english-ca

english-canada

english-gb

english-newzealand

english-nz

english-unitedkingdom

english-unitedstates

english-us

english

esperanto

estonian

ewe

ewondo

faroese

filipino

finnish

french-be

french-belgium

french-ca

french-canada

french-ch

french-lu

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

---

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

indonesian
interlingua
irish
italian
japanese
jolafonyi
kabuverdianu
kabyle
kako
kalaallisut
kalenjin
kamba
kannada
kashmiri
kazakh
khmer
kikuyu
kinyarwanda
konkani
korean
koyraborosenni
koyrachiini
kwasio
kyrgyz
lakota
langi
lao
latvian
lingala
lithuanian
lowersorbian
lsorbian
lubakatanga
luo
luxembourgish
luyia
macedonian
machame
makhuwameetto
makonde
malagasy
malay-bn
malay-brunei
malay-sg
malay-singapore
malay
malayalam
maltese
manx
marathi
masai
mazanderani
meru
meta

mexican
mongolian
morisyen
mundang
nama
nepali
newzealand
ngiemboon
ngomba
norsk
northernluri
northernsami
northndebele
norwegianbokmal
norwegiannynorsk
nswissgerman
nuer
nyankole
nynorsk
occitan
oriya
oromo
ossetic
pashto
persian
piedmontese
polish
polytonicgreek
portuguese-br
portuguese-brazil
portuguese-portugal
portuguese-pt
portuguese
punjabi-arab
punjabi-arabic
punjabi-gurmukhi
punjabi-guru
punjabi
quechua
romanian
romansh
rombo
rundi
russian
rwa
sakha
samburu
samin
sango
sangu
sanskrit-beng
sanskrit-bengali
sanskrit-deva
sanskrit-devanagari

sanskrit-gujarati
sanskrit-gujr
sanskrit-kannada
sanskrit-knda
sanskrit-malayalam
sanskrit-mlym
sanskrit-telu
sanskrit-telugu
sanskrit
scottishgaelic
sena
serbian-cyrillic-bosniaherzegovina
serbian-cyrillic-kosovo
serbian-cyrillic-montenegro
serbian-cyrillic
serbian-cyrl-ba
serbian-cyrl-me
serbian-cyrl-xk
serbian-cyrl
serbian-latin-bosniaherzegovina
serbian-latin-kosovo
serbian-latin-montenegro
serbian-latin
serbian-latn-ba
serbian-latn-me
serbian-latn-xk
serbian-latn
serbian
shambala
shona
sichuanyi
sinhala
slovak
slovene
slovenian
soga
somali
spanish-mexico
spanish-mx
spanish
standardmoroccantamazight
swahili
swedish
swissgerman
tachelhit-latin

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

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

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

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

1.14 Selecting fonts

New 3.15 Babel provides a high level interface on top of fontspec to select fonts. There is no need to load fontspec explicitly – babel does it for you with the first \babelfont.[14]

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

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

The main purpose of \babelfont is to define at once in a multilingual document the fonts required by the different languages, with their corresponding language systems (script and language). So, if you load, say, 4 languages, \babelfont{rm}{FreeSerif} defines 4 fonts (with their variants, of course), which are switched with the language by babel. It is a tool to make things easier and transparent to the user.
Here *font-family* is rm, sf or tt (or newly defined ones, as explained below), and *font-name* is the same as in fontspec and the like.
If no language is given, then it is considered the default font for the family, activated when a language is selected.
On the other hand, if there is one or more languages in the optional argument, the font will be assigned to them, overriding the default one. Alternatively, you may set a font for a script – just precede its name (lowercase) with a star (eg, *devanagari). With this optional argument, the font is *not* yet defined, but just predeclared. This means you may define as many fonts as you want 'just in case', because if the language is never selected, the corresponding \babelfont declaration is just ignored.
Babel takes care of the font language and the font script when languages are selected (as well as the writing direction); see the recognized languages above. In most cases, you will not need *font-options*, which is the same as in fontspec, but you may add further key/value pairs if necessary.

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

LUATEX/XETEX
```
\documentclass{article}

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

\babelprovide[import]{hebrew}

\babelfont{rm}{FreeSerif}

\begin{document}

Svenska \foreignlanguage{hebrew}{עִבְרִית} svenska.

\end{document}
```

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

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

---

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

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

EXAMPLE Here is how to do it:

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

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

NOTE You may load fontspec explicitly. For example:

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

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

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

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

- 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.

- 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}
```

- 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*⟩.

- 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.)

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.

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.

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: \mylangchaptername not set. Please, define it
(babel)                after the language has been loaded (typically
(babel)                in the preamble) with something like:
(babel)                \renewcommand\maylangchaptername{..}
(babel)                Reported on input line 18.
```

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}
\renewcommand\arhinishchaptername{Chapitula}
\renewcommand\arhinishrefname{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).

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}
```

script= ⟨*script-name*⟩

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

language= ⟨*language-name*⟩

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

alph= ⟨*counter-name*⟩

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

Alph= ⟨*counter-name*⟩

Same for \Alph.

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

onchar= ids | fonts

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

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

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

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

intrapenalty= ⟨*penalty*⟩

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

mapfont= `direction`

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

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

1.17 Digits and counters

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

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

Languages providing native digits in all or some variants are:

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

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

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

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

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

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

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

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

The styles are:

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

Russian `lower`, `lower.full`, `upper`, `upper.full`
Syriac `letters`
Tamil `ancient`
Thai `alphabetic`
Ukrainian `lower`, `lower.full`, `upper`, `upper.full`
Chinese `cjk-earthly-branch`, `cjk-heavenly-stem`, `fullwidth.lower.alpha`, `fullwidth.upper.alpha`

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

1.18   Dates

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

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

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

1.19   Accessing language info

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

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

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

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

\localeinfo   {⟨*field*⟩}

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

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

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

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

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

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

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

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

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

\localeid

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

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

1.20   Hyphenation and line breaking

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

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

New 3.9a It is customary to classify hyphens in two types: (1) *explicit* or *hard hyphens*, which in TeX are entered as -, and (2) *optional* or *soft hyphens*, which are entered as \-. Strictly, a *soft hyphen* is not a hyphen, but just a breaking opportunity or, in TeX terms, a "discretionary"; a *hard hyphen* is a hyphen with a breaking opportunity after it. A further type is a *non-breaking hyphen*, a hyphen without a breaking opportunity.

In TeX, - and \- forbid further breaking opportunities in the word. This is the desired behavior very often, but not always, and therefore many languages provide shorthands for these cases. Unfortunately, this has not been done consistently: for example, "- in Dutch, Portuguese, Catalan or Danish is a hard hyphen, while in German, Spanish, Norwegian, Slovak or Russian is a soft hyphen. Furthermore, some of them even redefine \-, so that you cannot insert a soft hyphen without breaking opportunities in the rest of the word. Therefore, some macros are provided with a set of basic "hyphens" which can be used by themselves, to define a user shorthand, or even in language files.

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

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

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

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

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

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

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

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

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

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

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

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

New 3.9m *In luatex only,*[15] adds or replaces patterns for the languages given or, without the optional argument, for *all* languages. If a pattern for a certain combination already exists, it gets replaced by the new one.

It can be used only in the preamble, and patterns are added when the language is first selected, thus taking into account changes of \lccodes's done in \extras⟨*lang*⟩ as well as the language-specific encoding (not set in the preamble by default). Multiple \babelpatterns's are allowed.

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

New 3.31 (Only luatex.) With \babelprovide and imported CJK languages, a simple generic line breaking algorithm (push-out-first) is applied, based on a selection of the Unicode rules (New 3.32 it is disabled in verbatim mode, or more precisely when the hyphenrules are set to nohyphenation). It can be activated alternatively by setting explicitly the intraspace.

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

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

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

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

This feature is activated with the first \babelposthyphenation.

See the babel wiki for a more detailed description and some examples. It also describes an additional replacement type with the key string.

EXAMPLE 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). For example, you can use the string replacement to 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:

---

[15]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.

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

In other words, it is a quite general tool. (A counterpart `\babelprehyphenation` is on the way.)

1.21   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.

1.22   Selecting scripts

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

\ensureascii   {⟨text⟩}

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

---

[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.

1.23 Selecting directions

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

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

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

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

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

There are some package options controlling bidi writing.

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

 New 3.14   Selects the bidi algorithm to be used. With `default` the bidi mechanism is just activated (by default it is not), but every change must be marked up. In xetex and pdftex this is the only option.
In luatex, `basic-r` provides a simple and fast method for R text, which handles numbers and unmarked L text within an R context many in typical cases. New 3.19   Finally, `basic` supports both L and R text, and it is the preferred method (support for `basic-r` is currently limited). (They are named `basic` mainly because they only consider the intrinsic direction of scripts and weak directionality.)
 New 3.29   In xetex, `bidi-r` and `bidi-l` resort to the package bidi (by Vafa Khalighi). Integration is still somewhat tentative, but it mostly works. For RL documents use the former, and for LR ones use the latter.
There are samples on GitHub, under `/required/babel/samples`. See particularly `lua-bidibasic.tex` and `lua-secenum.tex`.

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

```
\documentclass{article}

\usepackage[bidi=basic]{babel}

\babelprovide[import, main]{arabic}

\babelfont{rm}{FreeSerif}
```

```
\begin{document}

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

\end{document}
```

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

```
\documentclass{book}

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

\babelprovide[onchar=ids fonts]{arabic}

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

\begin{document}

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

\end{document}
```

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

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

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

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

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

New 3.16 *To be expanded.* Selects which layout elements are adapted in bidi documents, including some text elements (except with options loading the bidi package, which

42

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` (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 if you want sloped lines. It attempts to do the same for pgf/tikz. Somewhat experimental. New 3.32 .

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

EXAMPLE Typically, in an Arabic document you would need:

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

\babelsublr {⟨*lr-text*⟩}

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

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

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

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

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

\BabelPatchSection {⟨*section-name*⟩}

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

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

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

New 3.17 Something like:

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

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

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

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

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

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

---

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

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

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

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

1.24 Language attributes

\languageattribute

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

1.25 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).

1.26 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.

1.27   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).

---

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

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

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

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

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

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

1.28   Tweaking some features

`\babeladjust`	`{⟨key-value-list⟩}`

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

1.29   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:

48

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

(A recent version of inputenc is required.)

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

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

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

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

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

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

1.30 Current and future work

The current work is focused on the so-called complex scripts in luatex. In 8-bit engines, babel provided a basic support for bidi text as part of the style for Hebrew, but it is somewhat unsatisfactory and internally replaces some hardwired commands by other hardwired commands (generic changes would be much better).
Useful additions would be, for example, time, currency, addresses and personal names.[21]. But that is the easy part, because they don't require modifying the LaTeX internals. Calendars (Arabic, Persian, Indic, etc.) are under study.

---

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

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

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

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

1.31 Tentative and experimental code

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

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.

\babelprehyphenation

New 3.44   Note it is tentative, but the current behavior for glyphs should be correct. It is similar to `\babelposthyphenation`, but (as its name implies) applied before hyphenation. There are other differences: (1) the first argument is the locale instead the name of hyphenation patterns; (2) in the search patterns = has no special meaning (| is still reserved, but currently unused); (3) in the replacement, discretionaries are not accepted, only remove, , and string = ...

Currently it handles glyphs, not discretionaries or spaces (in particular, it will not catch the hyphen and you can't insert or remove spaces). Also, you are limited to substitutions as done by lua, although a future implementation may alternatively accept lpeg. Performance is still somewhat poor.

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]

---

[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`.

2.1 Format

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

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

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

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

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

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

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

A typical error when using babel is the following:

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

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

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.

---

[24]This is because different operating systems sometimes use *very* different file-naming conventions.
[25]This is not a new feature, but in former versions it didn't work correctly.

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

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

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

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

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

Some recommendations:

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

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

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

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

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

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

---

[26]But not removed, for backward compatibility.

3.1 Guidelines for contributed languages

Currently, the easiest way to contribute a new language is by taking one the the the 500 or so `ini` templates available on GitHub as a basis. Just make a pull request o dowonload it and then, after filling the fields, sent it to me. Fell free to ask for help or to make feature requests.

As to `ldf` files, now language files are "outsourced" and are located in a separate directory (`/macros/latex/contrib/babel-contrib`), so that they are contributed directly to CTAN (please, do not send to me language styles just to upload them to CTAN).

Of course, placing your style files in this directory is not mandatory, but if you want to do it, here are a few guidelines.

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

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

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

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

The following page provides a starting point for `ldf` files: `http://www.texnia.com/incubator.html`. See also `https://github.com/latex3/babel/wiki/List-of-locale-templates`.

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

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

3.5   Support for saving macro definitions

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

---

[27]This mechanism was introduced by Bernd Raichle.

56

\set@low@box is available. It takes one argument and puts that argument in an \hbox, at the baseline. The result is available in \box0 for further processing.

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

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

3.8   Encoding-dependent strings

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

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

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

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

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

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

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

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

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

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

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

57

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

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

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

\EndBabelCommands
```

A real example is:

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

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

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

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

\StartBabelCommands{german,austrian}{date}
  \SetString\monthiiname{Februar}
  \SetString\monthiiiname{M\"{a}rz}
  \SetString\monthivname{April}
  \SetString\monthvname{Mai}
  \SetString\monthviname{Juni}
  \SetString\monthviiname{Juli}
  \SetString\monthviiiname{August}
  \SetString\monthixname{September}
  \SetString\monthxname{Oktober}
  \SetString\monthxiname{November}
  \SetString\monthxiiname{Dezenber}
  \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).

---

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

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

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

`\EndBabelCommands`	Marks the end of the series of blocks.

`\AfterBabelCommands`	`{`⟨*code*⟩`}`

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

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

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

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

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

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

#1 is replaced by the roman numeral.

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

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

---

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

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

\EndBabelCommands
```

(Note the mapping for OT1 is not complete.)

\SetHyphenMap {⟨*to-lower-macros*⟩}

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

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

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

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

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

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

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

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=3.50.2159⟩⟩
2 ⟨⟨date=2020/10/13⟩⟩

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,
\bbl@afterfi   we take extra care to 'throw' it over the \else and \fi parts of an \if-statement[30]. These
macros will break if another \if...\fi statement appears in one of the arguments and it
is not enclosed in braces.

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

\bbl@exp   Now, just syntactical sugar, but it makes partial expansion of some code a lot more simple
and readable. Here \\ stands for \noexpand and \<..> for \noexpand applied to a built
macro name (the latter does not define the macro if undefined to \relax, because it is
created locally). The result may be followed by extra arguments, if necessary.

```
29 \def\bbl@exp#1{%
30    \begingroup
31       \let\\\noexpand
32       \def\<##1>{\expandafter\noexpand\csname##1\endcsname}%
33       \edef\bbl@exp@aux{\endgroup#1}%
34    \bbl@exp@aux}
```

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

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

---

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

```
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   {\providecommand\bbl@trace[1]{}%
198    \let\bbl@debug\@gobble}
199 ⟨⟨Basic macros⟩⟩
200   % Temporarily repeat here the code for errors
201   \def\bbl@error#1#2{%
202     \begingroup
203       \def\\{\MessageBreak}%
204       \PackageError{babel}{#1}{#2}%
205     \endgroup}
206   \def\bbl@warning#1{%
207     \begingroup
208       \def\\{\MessageBreak}%
209       \PackageWarning{babel}{#1}%
```

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

```
262 \ifx\bbl@languages\@undefined\else
263    \begingroup
264      \catcode`\^^I=12
265      \@ifpackagewith{babel}{showlanguages}{%
```

68

```
266      \begingroup
267        \def\bbl@elt#1#2#3#4{\wlog{#2^^I#1^^I#3^^I#4}}%
268        \wlog{<*languages>}%
269        \bbl@languages
270        \wlog{</languages>}%
271      \endgroup}{}
272    \endgroup
273    \def\bbl@elt#1#2#3#4{%
274      \ifnum#2=\z@
275        \gdef\bbl@nulllanguage{#1}%
276        \def\bbl@elt##1##2##3##4{}%
277      \fi}%
278    \bbl@languages
279 \fi%
```

7.3 base

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

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

```
280 \bbl@trace{Defining option 'base'}
281 \@ifpackagewith{babel}{base}{%
282   \let\bbl@onlyswitch\@empty
283   \let\bbl@provide@locale\relax
284   \input babel.def
285   \let\bbl@onlyswitch\@undefined
286   \ifx\directlua\@undefined
287     \DeclareOption*{\bbl@patterns{\CurrentOption}}%
288   \else
289     \input luababel.def
290     \DeclareOption*{\bbl@patterns@lua{\CurrentOption}}%
291   \fi
292   \DeclareOption{base}{}%
293   \DeclareOption{showlanguages}{}%
294   \ProcessOptions
295   \global\expandafter\let\csname opt@babel.sty\endcsname\relax
296   \global\expandafter\let\csname ver@babel.sty\endcsname\relax
297   \global\let\@ifl@ter@@\@ifl@ter
298   \def\@ifl@ter#1#2#3#4#5{\global\let\@ifl@ter\@ifl@ter@@}%
299   \endinput}{}%
300 % \end{macrocode}
301 %
302 % \subsection{\texttt{key=value} options and other general option}
303 %
304 %    The following macros extract language modifiers, and only real
305 %    package options are kept in the option list. Modifiers are saved
306 %    and assigned to |\BabelModifiers| at |\bbl@load@language|; when
307 %    no modifiers have been given, the former is |\relax|. How
308 %    modifiers are handled are left to language styles; they can use
309 %    |\in@|, loop them with |\@for| or load |keyval|, for example.
310 %
311 %    \begin{macrocode}
312 \bbl@trace{key=value and another general options}
313 \bbl@csarg\let{tempa\expandafter}\csname opt@babel.sty\endcsname
314 \def\bbl@tempb#1.#2{%  Remove trailing dot
```

```
315     #1\ifx\@empty#2\else,\bbl@afterfi\bbl@tempb#2\fi}%
316 \def\bbl@tempd#1.#2\@nnil{%  TODO. Refactor lists?
317   \ifx\@empty#2%
318     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
319   \else
320     \in@{,provide,}{,#1,}%
321     \ifin@
322       \edef\bbl@tempc{%
323         \ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.\bbl@tempb#2}%
324     \else
325       \in@{=}{#1}%
326       \ifin@
327         \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.#2}%
328       \else
329         \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
330         \bbl@csarg\edef{mod@#1}{\bbl@tempb#2}%
331       \fi
332     \fi
333   \fi}
334 \let\bbl@tempc\@empty
335 \bbl@foreach\bbl@tempa{\bbl@tempd#1.\@empty\@nnil}
336 \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.

```
337 \DeclareOption{KeepShorthandsActive}{}
338 \DeclareOption{activeacute}{}
339 \DeclareOption{activegrave}{}
340 \DeclareOption{debug}{}
341 \DeclareOption{noconfigs}{}
342 \DeclareOption{showlanguages}{}
343 \DeclareOption{silent}{}
344 \DeclareOption{mono}{}
345 \DeclareOption{shorthands=off}{\bbl@tempa shorthands=\bbl@tempa}
346 \chardef\bbl@iniflag\z@
347 \DeclareOption{provide=*}{\chardef\bbl@iniflag\@ne}     % main -> +1
348 \DeclareOption{provide+=*}{\chardef\bbl@iniflag\tw@}    % add = 2
349 \DeclareOption{provide*=*}{\chardef\bbl@iniflag\thr@@} % add + main
350 % Don't use. Experimental. TODO.
351 \newif\ifbbl@single
352 \DeclareOption{selectors=off}{\bbl@singletrue}
353 \let\bbl@autoload@options\@empty
354 % autoload with cat @=letter
355 \DeclareOption{provide@=*}{\def\bbl@autoload@options{import}}
356 \makeatother
357 \DeclareOption{provide@=*}% autoload with cat @=other
358   {\expandafter\def\csname bbl@autoload@options\endcsname{import}}
359 \makeatletter
360 ⟨⟨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.

```
361 \let\bbl@opt@shorthands\@nnil
```

```
362 \let\bbl@opt@config\@nnil
363 \let\bbl@opt@main\@nnil
364 \let\bbl@opt@headfoot\@nnil
365 \let\bbl@opt@layout\@nnil
```

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

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

```
377 \let\bbl@language@opts\@empty
378 \DeclareOption*{%
379   \bbl@xin@{\string=}{\CurrentOption}%
380   \ifin@
381     \expandafter\bbl@tempa\CurrentOption\bbl@tempa
382   \else
383     \bbl@add@list\bbl@language@opts{\CurrentOption}%
384   \fi}
```

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

```
385 \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=....

```
386 \bbl@trace{Conditional loading of shorthands}
387 \def\bbl@sh@string#1{%
388   \ifx#1\@empty\else
389     \ifx#1t\string~%
390     \else\ifx#1c\string,%
391     \else\string#1%
392     \fi\fi
393     \expandafter\bbl@sh@string
394   \fi}
395 \ifx\bbl@opt@shorthands\@nnil
396   \def\bbl@ifshorthand#1#2#3{#2}%
397 \else\ifx\bbl@opt@shorthands\@empty
398   \def\bbl@ifshorthand#1#2#3{#3}%
399 \else
```

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

```
400  \def\bbl@ifshorthand#1{%
401    \bbl@xin@{\string#1}{\bbl@opt@shorthands}%
402    \ifin@
403      \expandafter\@firstoftwo
404    \else
405      \expandafter\@secondoftwo
406    \fi}
```

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

```
407    \edef\bbl@opt@shorthands{%
408      \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.

```
409    \bbl@ifshorthand{'}%
410      {\PassOptionsToPackage{activeacute}{babel}}{}
411    \bbl@ifshorthand{`}%
412      {\PassOptionsToPackage{activegrave}{babel}}{}
413 \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.

```
414 \ifx\bbl@opt@headfoot\@nnil\else
415   \g@addto@macro\@resetactivechars{%
416     \set@typeset@protect
417     \expandafter\select@language@x\expandafter{\bbl@opt@headfoot}%
418     \let\protect\noexpand}
419 \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.

```
420 \ifx\bbl@opt@safe\@undefined
421   \def\bbl@opt@safe{BR}
422 \fi
423 \ifx\bbl@opt@main\@nnil\else
424   \edef\bbl@language@opts{%
425     \ifx\bbl@language@opts\@empty\else\bbl@language@opts,\fi
426       \bbl@opt@main}
427 \fi
```

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

```
428 \bbl@trace{Defining IfBabelLayout}
429 \ifx\bbl@opt@layout\@nnil
430   \newcommand\IfBabelLayout[3]{#3}%
431 \else
432   \newcommand\IfBabelLayout[1]{%
433     \@expandtwoargs\in@{.#1.}{.\bbl@opt@layout.}%
434     \ifin@
435       \expandafter\@firstoftwo
436     \else
437       \expandafter\@secondoftwo
438     \fi}
439 \fi
```

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

```
440 \input babel.def
```

7.5   Cross referencing macros

The LᴬTᴇX 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.

```
441 ⟨⟨*More package options⟩⟩ ≡
442 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
443 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
444 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
445 ⟨⟨/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.

```
446 \bbl@trace{Cross referencing macros}
447 \ifx\bbl@opt@safe\@empty\else
448   \def\@newl@bel#1#2#3{%
449     {\@safe@activestrue
450     \bbl@ifunset{#1@#2}%
451       \relax
452       {\gdef\@multiplelabels{%
453         \@latex@warning@no@line{There were multiply-defined labels}}%
454       \@latex@warning@no@line{Label `#2' multiply defined}}%
455     \global\@namedef{#1@#2}{#3}}}
```

\@testdef   An internal LᴬTᴇX macro used to test if the labels that have been written on the .aux file
have changed. It is called by the \enddocument macro.

```
456   \CheckCommand*\@testdef[3]{%
457     \def\reserved@a{#3}%
458     \expandafter\ifx\csname#1@#2\endcsname\reserved@a
459     \else
460       \@tempswatrue
461     \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.

```
462   \def\@testdef#1#2#3{%  TODO. With @samestring?
463     \@safe@activestrue
464     \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
465     \def\bbl@tempb{#3}%
466     \@safe@activesfalse
467     \ifx\bbl@tempa\relax
468     \else
469       \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
470     \fi
471     \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
```

```
472    \ifx\bbl@tempa\bbl@tempb
473    \else
474      \@tempswatrue
475    \fi}
476 \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.

```
477 \bbl@xin@{R}\bbl@opt@safe
478 \ifin@
479   \bbl@redefinerobust\ref#1{%
480     \@safe@activestrue\org@ref{#1}\@safe@activesfalse}
481   \bbl@redefinerobust\pageref#1{%
482     \@safe@activestrue\org@pageref{#1}\@safe@activesfalse}
483 \else
484   \let\org@ref\ref
485   \let\org@pageref\pageref
486 \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.

```
487 \bbl@xin@{B}\bbl@opt@safe
488 \ifin@
489   \bbl@redefine\@citex[#1]#2{%
490     \@safe@activestrue\edef\@tempa{#2}\@safe@activesfalse
491     \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.

```
492   \AtBeginDocument{%
493     \@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.)

```
494     \def\@citex[#1][#2]#3{%
495       \@safe@activestrue\edef\@tempa{#3}\@safe@activesfalse
496       \org@@citex[#1][#2]{\@tempa}}%
497     }{}}
```

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

```
498   \AtBeginDocument{%
499     \@ifpackageloaded{cite}{%
500       \def\@citex[#1]#2{%
501         \@safe@activestrue\org@@citex[#1]{#2}\@safe@activesfalse}%
502     }{}}
```

\nocite

The macro \nocite which is used to instruct BiBT$_E$X to extract uncited references from the database.

```
503    \bbl@redefine\nocite#1{%
504      \@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.

```
505    \bbl@redefine\bibcite{%
506      \bbl@cite@choice
507      \bibcite}
```

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

```
508    \def\bbl@bibcite#1#2{%
509      \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.

```
510    \def\bbl@cite@choice{%
511      \global\let\bibcite\bbl@bibcite
512      \@ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{}%
513      \@ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{}%
514      \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.

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

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

```
516    \bbl@redefine\@bibitem#1{%
517      \@safe@activestrue\org@@bibitem{#1}\@safe@activesfalse}
518 \else
519    \let\org@nocite\nocite
520    \let\org@@citex\@citex
521    \let\org@bibcite\bibcite
522    \let\org@@bibitem\@bibitem
523 \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.

```
524 \bbl@trace{Marks}
525 \IfBabelLayout{sectioning}
526    {\ifx\bbl@opt@headfoot\@nnil
527      \g@addto@macro\@resetactivechars{%
```

75

```
503    \bbl@redefine\nocite#1{%
504      \@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.

```
505    \bbl@redefine\bibcite{%
506      \bbl@cite@choice
507      \bibcite}
```

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

```
508    \def\bbl@bibcite#1#2{%
509      \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.

```
510    \def\bbl@cite@choice{%
511      \global\let\bibcite\bbl@bibcite
512      \@ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{}%
513      \@ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{}%
514      \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.

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

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

```
516    \bbl@redefine\@bibitem#1{%
517      \@safe@activestrue\org@@bibitem{#1}\@safe@activesfalse}
518 \else
519    \let\org@nocite\nocite
520    \let\org@@citex\@citex
521    \let\org@bibcite\bibcite
522    \let\org@@bibitem\@bibitem
523 \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.

```
524 \bbl@trace{Marks}
525 \IfBabelLayout{sectioning}
526    {\ifx\bbl@opt@headfoot\@nnil
527      \g@addto@macro\@resetactivechars{%
```

75

```
528        \set@typeset@protect
529        \expandafter\select@language@x\expandafter{\bbl@main@language}%
530        \let\protect\noexpand
531        \ifcase\bbl@bidimode\else % Only with bidi. See also above
532          \edef\thepage{%
533            \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
534        \fi}%
535    \fi}
536  {\ifbbl@single\else
537     \bbl@ifunset{markright }\bbl@redefine\bbl@redefinerobust
538     \markright#1{%
539       \bbl@ifblank{#1}%
540         {\org@markright{}}%
541         {\toks@{#1}%
542          \bbl@exp{%
543            \\\org@markright{\\\protect\\\foreignlanguage{\languagename}%
544              {\\\protect\\\bbl@restore@actives\the\toks@}}}}}%
```

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

```
545        \ifx\@mkboth\markboth
546          \def\bbl@tempc{\let\@mkboth\markboth}
547        \else
548          \def\bbl@tempc{}
549        \fi
550        \bbl@ifunset{markboth }\bbl@redefine\bbl@redefinerobust
551        \markboth#1#2{%
552          \protected@edef\bbl@tempb##1{%
553            \protect\foreignlanguage
554            {\languagename}{\protect\bbl@restore@actives##1}}%
555          \bbl@ifblank{#1}%
556            {\toks@{}}%
557            {\toks@\expandafter{\bbl@tempb{#1}}}%
558          \bbl@ifblank{#2}%
559            {\@temptokena{}}%
560            {\@temptokena\expandafter{\bbl@tempb{#2}}}%
561          \bbl@exp{\\\org@markboth{\the\toks@}{\the\@temptokena}}}
562          \bbl@tempc
563      \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.

```
564 \bbl@trace{Preventing clashes with other packages}
565 \bbl@xin@{R}\bbl@opt@safe
566 \ifin@
567   \AtBeginDocument{%
568     \@ifpackageloaded{ifthen}{%
569       \bbl@redefine@long\ifthenelse#1#2#3{%
570         \let\bbl@temp@pref\pageref
571         \let\pageref\org@pageref
572         \let\bbl@temp@ref\ref
573         \let\ref\org@ref
574         \@safe@activestrue
575         \org@ifthenelse{#1}%
576           {\let\pageref\bbl@temp@pref
577            \let\ref\bbl@temp@ref
578            \@safe@activesfalse
579            #2}%
580           {\let\pageref\bbl@temp@pref
581            \let\ref\bbl@temp@ref
582            \@safe@activesfalse
583            #3}%
584       }%
585     }{}%
586   }
```

7.7.2 varioref

\@@vpageref
\vrefpagenum
\Ref

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

```
587   \AtBeginDocument{%
588     \@ifpackageloaded{varioref}{%
589       \bbl@redefine\@@vpageref#1[#2]#3{%
590         \@safe@activestrue
591         \org@@@vpageref{#1}[#2]{#3}%
592         \@safe@activesfalse}%
593       \bbl@redefine\vrefpagenum#1#2{%
594         \@safe@activestrue
595         \org@vrefpagenum{#1}{#2}%
596         \@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.

```
597       \expandafter\def\csname Ref \endcsname#1{%
598         \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
599     }{}%
600   }
601 \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.

```
602 \AtEndOfPackage{%
603   \AtBeginDocument{%
604     \@ifpackageloaded{hhline}%
605       {\expandafter\ifx\csname normal@char\string:\endcsname\relax
606         \else
607           \makeatletter
608           \def\@currname{hhline}\input{hhline.sty}\makeatother
609       \fi}%
610       {}}}
```

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.

```
611 % \AtBeginDocument{%
612 %   \ifx\pdfstringdefDisableCommands\@undefined\else
613 %     \pdfstringdefDisableCommands{\languageshorthands{system}}%
614 %   \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.

```
615 \DeclareRobustCommand{\FOREIGNLANGUAGE}[1]{%
616   \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.

```
617 \def\substitutefontfamily#1#2#3{%
618   \lowercase{\immediate\openout15=#1#2.fd\relax}%
619   \immediate\write15{%
620     \string\ProvidesFile{#1#2.fd}%
621     [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
622      \space generated font description file]^^J
623     \string\DeclareFontFamily{#1}{#2}{}^^J
624     \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{}^^J
625     \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{}^^J
626     \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{}^^J
627     \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{}^^J
628     \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{}^^J
629     \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{}^^J
630     \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{}^^J
631     \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{}^^J
```

```
632      }%
633    \closeout15
634    }
635 \@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.

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

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

79

current encoding at the end of processing the package is the Latin encoding.

```
672 \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.

```
673 \AtBeginDocument{%
674   \@ifpackageloaded{fontspec}%
675     {\xdef\latinencoding{%
676        \ifx\UTFencname\@undefined
677          EU\ifcase\bbl@engine\or2\or1\fi
678        \else
679          \UTFencname
680        \fi}}%
681     {\gdef\latinencoding{OT1}%
682      \ifx\cf@encoding\bbl@t@one
683        \xdef\latinencoding{\bbl@t@one}%
684      \else
685        \ifx\@fontenc@load@list\@undefined
686          \@ifl@aded{def}{t1enc}{\xdef\latinencoding{\bbl@t@one}}{}%
687        \else
688          \def\@elt#1{,#1,}%
689          \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
690          \let\@elt\relax
691          \bbl@xin@{,T1,}\bbl@tempa
692          \ifin@
693            \xdef\latinencoding{\bbl@t@one}%
694          \fi
695        \fi
696      \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.

```
697 \DeclareRobustCommand{\latintext}{%
698   \fontencoding{\latinencoding}\selectfont
699   \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.

```
700 \ifx\@undefined\DeclareTextFontCommand
701   \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
702 \else
703   \DeclareTextFontCommand{\textlatin}{\latintext}
704 \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.

```
705 \ifodd\bbl@engine
706   \def\bbl@activate@preotf{%
707     \let\bbl@activate@preotf\relax  % only once
708     \directlua{
709       Babel = Babel or {}
710       %
711       function Babel.pre_otfload_v(head)
712         if Babel.numbers and Babel.digits_mapped then
713           head = Babel.numbers(head)
714         end
715         if Babel.bidi_enabled then
716           head = Babel.bidi(head, false, dir)
717         end
718         return head
719       end
720       %
721       function Babel.pre_otfload_h(head, gc, sz, pt, dir)
722         if Babel.numbers and Babel.digits_mapped then
723           head = Babel.numbers(head)
724         end
725         if Babel.bidi_enabled then
726           head = Babel.bidi(head, false, dir)
727         end
728         return head
729       end
730       %
731       luatexbase.add_to_callback('pre_linebreak_filter',
732         Babel.pre_otfload_v,
733         'Babel.pre_otfload_v',
734         luatexbase.priority_in_callback('pre_linebreak_filter',
735           'luaotfload.node_processor') or nil)
736       %
737       luatexbase.add_to_callback('hpack_filter',
738         Babel.pre_otfload_h,
739         'Babel.pre_otfload_h',
740         luatexbase.priority_in_callback('hpack_filter',
741           'luaotfload.node_processor') or nil)
742     }}
743 \fi
```

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

```
744 \bbl@trace{Loading basic (internal) bidi support}
745 \ifodd\bbl@engine
746   \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
747     \let\bbl@beforeforeign\leavevmode
748     \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
749     \RequirePackage{luatexbase}
750     \bbl@activate@preotf
751     \directlua{
752       require('babel-data-bidi.lua')
753       \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
754         require('babel-bidi-basic.lua')
755       \or
756         require('babel-bidi-basic-r.lua')
757       \fi}
758     % TODO - to locale_props, not as separate attribute
759     \newattribute\bbl@attr@dir
760     % TODO. I don't like it, hackish:
761     \bbl@exp{\output{\bodydir\pagedir\the\output}}
762     \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
763   \fi\fi
764 \else
765   \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
766     \bbl@error
767       {The bidi method `basic' is available only in\\%
768        luatex. I'll continue with `bidi=default', so\\%
769        expect wrong results}%
770       {See the manual for further details.}%
771     \let\bbl@beforeforeign\leavevmode
772     \AtEndOfPackage{%
773       \EnableBabelHook{babel-bidi}%
774       \bbl@xebidipar}
775   \fi\fi
776   \def\bbl@loadxebidi#1{%
777     \ifx\RTLfootnotetext\@undefined
778       \AtEndOfPackage{%
779         \EnableBabelHook{babel-bidi}%
780         \ifx\fontspec\@undefined
781           \bbl@loadfontspec % bidi needs fontspec
782         \fi
783         \usepackage#1{bidi}}%
784     \fi}
785   \ifnum\bbl@bidimode>200
786     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
787       \bbl@tentative{bidi=bidi}
788       \bbl@loadxebidi{}
789     \or
790       \bbl@loadxebidi{[rldocument]}
791     \or
792       \bbl@loadxebidi{}
793     \fi
794   \fi
795 \fi
796 \ifnum\bbl@bidimode=\@ne
797   \let\bbl@beforeforeign\leavevmode
798   \ifodd\bbl@engine
799     \newattribute\bbl@attr@dir
```

```
800       \bbl@exp{\output{\bodydir\pagedir\the\output}}%
801     \fi
802     \AtEndOfPackage{%
803       \EnableBabelHook{babel-bidi}%
804       \ifodd\bbl@engine\else
805         \bbl@xebidipar
806       \fi}
807 \fi
```

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

```
808 \bbl@trace{Macros to switch the text direction}
809 \def\bbl@alscripts{,Arabic,Syriac,Thaana,}
810 \def\bbl@rscripts{% TODO. Base on codes ??
811     ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
812     Old Hungarian,Old Hungarian,Lydian,Mandaean,Manichaean,%
813     Manichaean,Meroitic Cursive,Meroitic,Old North Arabian,%
814     Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
815     Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
816     Old South Arabian,}%
817 \def\bbl@provide@dirs#1{%
818     \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
819     \ifin@
820       \global\bbl@csarg\chardef{wdir@#1}\@ne
821       \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
822       \ifin@
823         \global\bbl@csarg\chardef{wdir@#1}\tw@  % useless in xetex
824       \fi
825     \else
826       \global\bbl@csarg\chardef{wdir@#1}\z@
827     \fi
828     \ifodd\bbl@engine
829       \bbl@csarg\ifcase{wdir@#1}%
830         \directlua{ Babel.locale_props[\the\localeid].textdir = 'l' }%
831       \or
832         \directlua{ Babel.locale_props[\the\localeid].textdir = 'r' }%
833       \or
834         \directlua{ Babel.locale_props[\the\localeid].textdir = 'al' }%
835       \fi
836   \fi}
837 \def\bbl@switchdir{%
838     \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
839     \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
840     \bbl@exp{\\\bbl@setdirs\bbl@cl{wdir}}}
841 \def\bbl@setdirs#1{% TODO - math
842     \ifcase\bbl@select@type % TODO - strictly, not the right test
843       \bbl@bodydir{#1}%
844       \bbl@pardir{#1}%
845     \fi
846   \bbl@textdir{#1}}
847 % TODO. Only if \bbl@bidimode > 0?:
848 \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
849 \DisableBabelHook{babel-bidi}
```

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

```
850 \ifodd\bbl@engine  % luatex=1
851     \chardef\bbl@thetextdir\z@
852     \chardef\bbl@thepardir\z@
853     \def\bbl@getluadir#1{%
```

```
854    \directlua{
855      if tex.#1dir == 'TLT' then
856        tex.sprint('0')
857      elseif tex.#1dir == 'TRT' then
858        tex.sprint('1')
859      end}}
860  \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
861    \ifcase#3\relax
862      \ifcase\bbl@getluadir{#1}\relax\else
863        #2 TLT\relax
864      \fi
865    \else
866      \ifcase\bbl@getluadir{#1}\relax
867        #2 TRT\relax
868      \fi
869    \fi}
870  \def\bbl@textdir#1{%
871    \bbl@setluadir{text}\textdir{#1}%
872    \chardef\bbl@thetextdir#1\relax
873    \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*3+#1}}
874  \def\bbl@pardir#1{%
875    \bbl@setluadir{par}\pardir{#1}%
876    \chardef\bbl@thepardir#1\relax}
877  \def\bbl@bodydir{\bbl@setluadir{body}\bodydir}
878  \def\bbl@pagedir{\bbl@setluadir{page}\pagedir}
879  \def\bbl@dirparastext{\pardir\the\textdir\relax}%    %%%%
880  % Sadly, we have to deal with boxes in math with basic.
881  % Activated every math with the package option bidi=:
882  \def\bbl@mathboxdir{%
883    \ifcase\bbl@thetextdir\relax
884      \everyhbox{\textdir TLT\relax}%
885    \else
886      \everyhbox{\textdir TRT\relax}%
887    \fi}
888  \frozen@everymath\expandafter{%
889    \expandafter\bbl@mathboxdir\the\frozen@everymath}
890  \frozen@everydisplay\expandafter{%
891    \expandafter\bbl@mathboxdir\the\frozen@everydisplay}
892 \else % pdftex=0, xetex=2
893    \newcount\bbl@dirlevel
894    \chardef\bbl@thetextdir\z@
895    \chardef\bbl@thepardir\z@
896    \def\bbl@textdir#1{%
897      \ifcase#1\relax
898        \chardef\bbl@thetextdir\z@
899        \bbl@textdir@i\beginL\endL
900      \else
901        \chardef\bbl@thetextdir\@ne
902        \bbl@textdir@i\beginR\endR
903      \fi}
904    \def\bbl@textdir@i#1#2{%
905      \ifhmode
906        \ifnum\currentgrouplevel>\z@
907          \ifnum\currentgrouplevel=\bbl@dirlevel
908            \bbl@error{Multiple bidi settings inside a group}%
909              {I'll insert a new group, but expect wrong results.}%
910            \bgroup\aftergroup#2\aftergroup\egroup
911          \else
912            \ifcase\currentgrouptype\or % 0 bottom
```

```
913            \aftergroup#2% 1 simple {}
914          \or
915            \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
916          \or
917            \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
918          \or\or\or % vbox vtop align
919          \or
920            \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
921          \or\or\or\or\or\or % output math disc insert vcent mathchoice
922          \or
923            \aftergroup#2% 14 \begingroup
924          \else
925            \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
926          \fi
927        \fi
928        \bbl@dirlevel\currentgrouplevel
929      \fi
930      #1%
931    \fi}
932    \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
933    \let\bbl@bodydir\@gobble
934    \let\bbl@pagedir\@gobble
935    \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).

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

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

```
962 \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}
963 \AtBeginDocument{%
964   \ifx\pdfstringdefDisableCommands\@undefined\else
965     \ifx\pdfstringdefDisableCommands\relax\else
```

```
966        \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
967      \fi
968    \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.

```
969 \bbl@trace{Local Language Configuration}
970 \ifx\loadlocalcfg\@undefined
971   \@ifpackagewith{babel}{noconfigs}%
972     {\let\loadlocalcfg\@gobble}%
973     {\def\loadlocalcfg#1{%
974        \InputIfFileExists{#1.cfg}%
975          {\typeout{*************************************^^J%
976                        * Local config file #1.cfg used^^J%
977                        *}}%
978          \@empty}}
979 \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?

```
980 \ifx\@unexpandable@protect\@undefined
981   \def\@unexpandable@protect{\noexpand\protect\noexpand}
982   \long\def\protected@write#1#2#3{%
983     \begingroup
984       \let\thepage\relax
985       #2%
986       \let\protect\@unexpandable@protect
987       \edef\reserved@a{\write#1{#3}}%
988       \reserved@a
989     \endgroup
990     \if@nobreak\ifvmode\nobreak\fi\fi}
991 \fi
992 %
993 % \subsection{Language options}
994 %
995 % Languages are loaded when processing the corresponding option
996 % \textit{except} if a |main| language has been set. In such a
997 % case, it is not loaded until all options has been processed.
998 % The following macro inputs the ldf file and does some additional
999 % checks (|\input| works, too, but possible errors are not catched).
1000 %
1001 %    \begin{macrocode}
1002 \bbl@trace{Language options}
1003 \let\bbl@afterlang\relax
1004 \let\BabelModifiers\relax
1005 \let\bbl@loaded\@empty
1006 \def\bbl@load@language#1{%
1007   \InputIfFileExists{#1.ldf}%
1008     {\edef\bbl@loaded{\CurrentOption
1009        \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}%
1010     \expandafter\let\expandafter\bbl@afterlang
```

```
1011         \csname\CurrentOption.ldf-h@@k\endcsname
1012       \expandafter\let\expandafter\BabelModifiers
1013           \csname bbl@mod@\CurrentOption\endcsname}%
1014     {\bbl@error{%
1015       Unknown option `\CurrentOption'. Either you misspelled it\\%
1016       or the language definition file \CurrentOption.ldf was not found}{%
1017       Valid options are, among others: shorthands=, KeepShorthandsActive,\\%
1018       activeacute, activegrave, noconfigs, safe=, main=, math=\\%
1019       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.

```
1020 \def\bbl@try@load@lang#1#2#3{%
1021   \IfFileExists{\CurrentOption.ldf}%
1022     {\bbl@load@language{\CurrentOption}}%
1023     {#1\bbl@load@language{#2}#3}}
1024 \DeclareOption{afrikaans}{\bbl@try@load@lang{}{dutch}{}}
1025 \DeclareOption{hebrew}{%
1026   \input{rlbabel.def}%
1027   \bbl@load@language{hebrew}}
1028 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}{}}
1029 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}{}}
1030 \DeclareOption{nynorsk}{\bbl@try@load@lang{}{norsk}{}}
1031 \DeclareOption{polutonikogreek}{%
1032   \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}
1033 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}{}}
1034 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}{}}
1035 \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.

```
1036 \ifx\bbl@opt@config\@nnil
1037   \@ifpackagewith{babel}{noconfigs}{}%
1038     {\InputIfFileExists{bblopts.cfg}%
1039       {\typeout{***********************************^^J%
1040               * Local config file bblopts.cfg used^^J%
1041               *}}%
1042       {}}%
1043 \else
1044   \InputIfFileExists{\bbl@opt@config.cfg}%
1045     {\typeout{***********************************^^J%
1046             * Local config file \bbl@opt@config.cfg used^^J%
1047             *}}%
1048     {\bbl@error{%
1049       Local config file `\bbl@opt@config.cfg' not found}{%
1050       Perhaps you misspelled it.}}%
1051 \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.

```
1052 \let\bbl@tempc\relax
```

```
1053 \bbl@foreach\bbl@language@opts{%
1054   \ifcase\bbl@iniflag
1055     \bbl@ifunset{ds@#1}%
1056       {\DeclareOption{#1}{\bbl@load@language{#1}}}%
1057       {}%
1058   \or
1059     \@gobble % case 2 same as 1
1060   \or
1061     \bbl@ifunset{ds@#1}%
1062       {\IfFileExists{#1.ldf}{}%
1063         {\IfFileExists{babel-#1.tex}{}{\DeclareOption{#1}{}}}}%
1064       {}%
1065     \bbl@ifunset{ds@#1}%
1066       {\def\bbl@tempc{#1}%
1067        \DeclareOption{#1}{%
1068          \ifnum\bbl@iniflag>\@ne
1069            \bbl@ldfinit
1070            \babelprovide[import]{#1}%
1071            \bbl@afterldf{}%
1072          \else
1073            \bbl@load@language{#1}%
1074          \fi}}%
1075       {}%
1076   \or
1077     \def\bbl@tempc{#1}%
1078     \bbl@ifunset{ds@#1}%
1079       {\DeclareOption{#1}{%
1080          \bbl@ldfinit
1081          \babelprovide[import]{#1}%
1082          \bbl@afterldf{}}}%
1083       {}%
1084   \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.

```
1085 \let\bbl@tempb\@nnil
1086 \bbl@foreach\@classoptionslist{%
1087   \bbl@ifunset{ds@#1}%
1088     {\IfFileExists{#1.ldf}{}%
1089       {\IfFileExists{babel-#1.tex}{}{\DeclareOption{#1}{}}}}%
1090     {}%
1091   \bbl@ifunset{ds@#1}%
1092     {\def\bbl@tempb{#1}%
1093      \DeclareOption{#1}{%
1094        \ifnum\bbl@iniflag>\@ne
1095          \bbl@ldfinit
1096          \babelprovide[import]{#1}%
1097          \bbl@afterldf{}%
1098        \else
1099          \bbl@load@language{#1}%
1100        \fi}}%
1101     {}}
```

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

```
1102 \ifnum\bbl@iniflag=\z@\else
1103   \ifx\bbl@opt@main\@nnil
1104     \ifx\bbl@tempc\relax
1105       \let\bbl@opt@main\bbl@tempb
```

```
1106      \else
1107        \let\bbl@opt@main\bbl@tempc
1108      \fi
1109    \fi
1110 \fi
1111 \ifx\bbl@opt@main\@nnil\else
1112    \expandafter
1113    \let\expandafter\bbl@loadmain\csname ds@\bbl@opt@main\endcsname
1114    \expandafter\let\csname ds@\bbl@opt@main\endcsname\@empty
1115 \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):

```
1116 \def\AfterBabelLanguage#1{%
1117    \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang}{}}
1118 \DeclareOption*{}
1119 \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.

```
1120 \bbl@trace{Option 'main'}
1121 \ifx\bbl@opt@main\@nnil
1122    \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}
1123    \let\bbl@tempc\@empty
1124    \bbl@for\bbl@tempb\bbl@tempa{%
1125      \bbl@xin@{,\bbl@tempb,}{,\bbl@loaded,}%
1126      \ifin@\edef\bbl@tempc{\bbl@tempb}\fi}
1127    \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
1128    \expandafter\bbl@tempa\bbl@loaded,\@nnil
1129    \ifx\bbl@tempb\bbl@tempc\else
1130      \bbl@warning{%
1131        Last declared language option is `\bbl@tempc',\\%
1132        but the last processed one was `\bbl@tempb'.\\%
1133        The main language cannot be set as both a global\\%
1134        and a package option. Use `main=\bbl@tempc' as\\%
1135        option. Reported}%
1136    \fi
1137 \else
1138    \ifodd\bbl@iniflag  % case 1,3
1139      \bbl@ldfinit
1140      \let\CurrentOption\bbl@opt@main
1141      \bbl@exp{\\\babelprovide[import,main]{\bbl@opt@main}}
1142      \bbl@afterldf{}%
1143    \else % case 0,2
1144      \chardef\bbl@iniflag\z@  % Force ldf
1145      \expandafter\let\csname ds@\bbl@opt@main\endcsname\bbl@loadmain
1146      \ExecuteOptions{\bbl@opt@main}
1147      \DeclareOption*{}%
1148      \ProcessOptions*
1149    \fi
1150 \fi
1151 \def\AfterBabelLanguage{%
1152    \bbl@error
```

89

```
1153        {Too late for \string\AfterBabelLanguage}%
1154        {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.

```
1155 \ifx\bbl@main@language\@undefined
1156   \bbl@info{%
1157     You haven't specified a language. I'll use 'nil'\\%
1158     as the main language. Reported}
1159     \bbl@load@language{nil}
1160 \fi
1161 ⟨/package⟩
1162 ⟨∗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

```
1163 \ifx\ldf@quit\@undefined\else
1164 \endinput\fi % Same line!
1165 ⟨⟨Make sure ProvidesFile is defined⟩⟩
1166 \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).

```
1167 \ifx\AtBeginDocument\@undefined  % TODO. change test.
1168   ⟨⟨Emulate LaTeX⟩⟩
1169   \def\languagename{english}%
1170   \let\bbl@opt@shorthands\@nnil
1171   \def\bbl@ifshorthand#1#2#3{#2}%
1172   \let\bbl@language@opts\@empty
1173   \ifx\babeloptionstrings\@undefined
1174     \let\bbl@opt@strings\@nnil
1175   \else
1176     \let\bbl@opt@strings\babeloptionstrings
1177   \fi
1178   \def\BabelStringsDefault{generic}
1179   \def\bbl@tempa{normal}
1180   \ifx\babeloptionmath\bbl@tempa
1181     \def\bbl@mathnormal{\noexpand\textormath}
```

```
1182    \fi
1183    \def\AfterBabelLanguage#1#2{}
1184    \ifx\BabelModifiers\@undefined\let\BabelModifiers\relax\fi
1185    \let\bbl@afterlang\relax
1186    \def\bbl@opt@safe{BR}
1187    \ifx\@uclclist\@undefined\let\@uclclist\@empty\fi
1188    \ifx\bbl@trace\@undefined\def\bbl@trace#1{}\fi
1189    \expandafter\newif\csname ifbbl@single\endcsname
1190    \chardef\bbl@bidimode\z@
1191 \fi
```

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

```
1192 \ifx\bbl@trace\@undefined
1193    \let\LdfInit\endinput
1194    \def\ProvidesLanguage#1{\endinput}
1195 \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.

1196 ⟨⟨*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.

```
1197 \def\bbl@version{⟨⟨version⟩⟩}
1198 \def\bbl@date{⟨⟨date⟩⟩}
1199 \def\adddialect#1#2{%
1200    \global\chardef#1#2\relax
1201    \bbl@usehooks{adddialect}{{#1}{#2}}%
1202    \begingroup
1203      \count@#1\relax
1204      \def\bbl@elt##1##2##3##4{%
1205        \ifnum\count@=##2\relax
1206          \bbl@info{\string#1 = using hyphenrules for ##1\\%
1207                    (\string\language\the\count@)}%
1208          \def\bbl@elt####1####2####3####4{}%
1209        \fi}%
1210      \bbl@cs{languages}%
1211    \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.

```
1212 \def\bbl@fixname#1{%
1213    \begingroup
1214      \def\bbl@tempe{l@}%
1215      \edef\bbl@tempd{\noexpand\@ifundefined{\noexpand\bbl@tempe#1}}%
1216      \bbl@tempd
```

91

```
1217        {\lowercase\expandafter{\bbl@tempd}%
1218          {\uppercase\expandafter{\bbl@tempd}%
1219            \@empty
1220            {\edef\bbl@tempd{\def\noexpand#1{#1}}%
1221             \uppercase\expandafter{\bbl@tempd}}}%
1222          {\edef\bbl@tempd{\def\noexpand#1{#1}}%
1223           \lowercase\expandafter{\bbl@tempd}}}%
1224        \@empty
1225      \edef\bbl@tempd{\endgroup\def\noexpand#1{#1}}%
1226    \bbl@tempd
1227    \bbl@exp{\\\bbl@usehooks{languagename}{{\languagename}{#1}}}}}
1228 \def\bbl@iflanguage#1{%
1229    \@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.

```
1230 \def\bbl@bcpcase#1#2#3#4\@@#5{%
1231    \ifx\@empty#3%
1232      \uppercase{\def#5{#1#2}}%
1233    \else
1234      \uppercase{\def#5{#1}}%
1235      \lowercase{\edef#5{#5#2#3#4}}%
1236    \fi}
1237 \def\bbl@bcplookup#1-#2-#3-#4\@@{%
1238    \let\bbl@bcp\relax
1239    \lowercase{\def\bbl@tempa{#1}}%
1240    \ifx\@empty#2%
1241      \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
1242    \else\ifx\@empty#3%
1243      \bbl@bcpcase#2\@empty\@empty\@@\bbl@tempb
1244      \IfFileExists{babel-\bbl@tempa-\bbl@tempb.ini}%
1245        {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb}}%
1246        {}%
1247      \ifx\bbl@bcp\relax
1248        \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
1249      \fi
1250    \else
1251      \bbl@bcpcase#2\@empty\@empty\@@\bbl@tempb
1252      \bbl@bcpcase#3\@empty\@empty\@@\bbl@tempc
1253      \IfFileExists{babel-\bbl@tempa-\bbl@tempb-\bbl@tempc.ini}%
1254        {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb-\bbl@tempc}}%
1255        {}%
1256      \ifx\bbl@bcp\relax
1257        \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
1258          {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
1259          {}%
1260      \fi
1261      \ifx\bbl@bcp\relax
1262        \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
1263          {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
1264          {}%
1265      \fi
1266      \ifx\bbl@bcp\relax
```

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

```
1304 \def\iflanguage#1{%
1305   \bbl@iflanguage{#1}{%
1306     \ifnum\csname l@#1\endcsname=\language
1307       \expandafter\@firstoftwo
1308     \else
1309       \expandafter\@secondoftwo
1310     \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.

```
1311 \let\bbl@select@type\z@
```

```
1312 \edef\selectlanguage{%
1313   \noexpand\protect
1314   \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.

```
1315 \ifx\@undefined\protect\let\protect\relax\fi
```

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

```
1316 \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.

```
1317 \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
\bbl@pop@language be simple:

```
1318 \def\bbl@push@language{%
1319   \ifx\languagename\@undefined\else
1320     \xdef\bbl@language@stack{\languagename+\bbl@language@stack}%
1321   \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.

```
1322 \def\bbl@pop@lang#1+#2\@@{%
1323   \edef\languagename{#1}%
1324   \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).

```
1325 \let\bbl@ifrestoring\@secondoftwo
1326 \def\bbl@pop@language{%
1327   \expandafter\bbl@pop@lang\bbl@language@stack\@@
```

```
1328    \let\bbl@ifrestoring\@firstoftwo
1329    \expandafter\bbl@set@language\expandafter{\languagename}%
1330    \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).

```
1331 \chardef\localeid\z@
1332 \def\bbl@id@last{0}     % No real need for a new counter
1333 \def\bbl@id@assign{%
1334   \bbl@ifunset{bbl@id@@\languagename}%
1335     {\count@\bbl@id@last\relax
1336      \advance\count@\@ne
1337      \bbl@csarg\chardef{id@@\languagename}\count@
1338      \edef\bbl@id@last{\the\count@}%
1339      \ifcase\bbl@engine\or
1340        \directlua{
1341          Babel = Babel or {}
1342          Babel.locale_props = Babel.locale_props or {}
1343          Babel.locale_props[bbl@id@last] = {}
1344          Babel.locale_props[bbl@id@last].name = '\languagename'
1345        }%
1346      \fi}%
1347     {}%
1348    \chardef\localeid\bbl@cl{id@}}
```

The unprotected part of \selectlanguage.

```
1349 \expandafter\def\csname selectlanguage \endcsname#1{%
1350   \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\tw@\fi
1351   \bbl@push@language
1352   \aftergroup\bbl@pop@language
1353   \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.

```
1354 \def\BabelContentsFiles{toc,lof,lot}
1355 \def\bbl@set@language#1{% from selectlanguage, pop@
1356   % The old buggy way. Preserved for compatibility.
1357   \edef\languagename{%
1358     \ifnum\escapechar=\expandafter`\string#1\@empty
1359     \else\string#1\@empty\fi}%
1360   \ifcat\relax\noexpand#1%
1361     \expandafter\ifx\csname date\languagename\endcsname\relax
1362       \edef\languagename{#1}%
1363       \let\localename\languagename
1364     \else
1365       \bbl@info{Using '\string\language' instead of 'language' is\\%
1366                 deprecated. If what you want is to use a\\%
```

```
1367                  macro containing the actual locale, make\\%
1368                  sure it does not not match any language.\\%
1369                  Reported}%
1370 %                    I'll\\%
1371 %                    try to fix '\string\localename', but I cannot promise\\%
1372 %                    anything. Reported}%
1373       \ifx\scantokens\@undefined
1374         \def\localename{??}%
1375       \else
1376         \scantokens\expandafter{\expandafter
1377           \def\expandafter\localename\expandafter{\languagename}}%
1378       \fi
1379     \fi
1380   \else
1381     \def\localename{#1}% This one has the correct catcodes
1382   \fi
1383   \select@language{\languagename}%
1384   % write to auxs
1385   \expandafter\ifx\csname date\languagename\endcsname\relax\else
1386     \if@filesw
1387       \ifx\babel@aux\@gobbletwo\else % Set if single in the first, redundant
1388         \protected@write\@auxout{}{\string\babel@aux{\bbl@auxname}{}}%
1389       \fi
1390       \bbl@usehooks{write}{}%
1391     \fi
1392   \fi}
1393 %
1394 \newif\ifbbl@bcpallowed
1395 \bbl@bcpallowedfalse
1396 \def\select@language#1{% from set@, babel@aux
1397   % set hymap
1398   \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
1399   % set name
1400   \edef\languagename{#1}%
1401   \bbl@fixname\languagename
1402   % TODO. name@map must be here?
1403   \bbl@provide@locale
1404   \bbl@iflanguage\languagename{%
1405       \expandafter\ifx\csname date\languagename\endcsname\relax
1406       \bbl@error
1407         {Unknown language `\languagename'. Either you have\\%
1408          misspelled its name, it has not been installed,\\%
1409          or you requested it in a previous run. Fix its name,\\%
1410          install it or just rerun the file, respectively. In\\%
1411          some cases, you may need to remove the aux file}%
1412         {You may proceed, but expect wrong results}%
1413     \else
1414       % set type
1415       \let\bbl@select@type\z@
1416       \expandafter\bbl@switch\expandafter{\languagename}%
1417     \fi}}
1418 \def\babel@aux#1#2{% TODO. See how to avoid undefined nil's
1419   \select@language{#1}%
1420   \bbl@foreach\BabelContentsFiles{%
1421     \@writefile{##1}{\babel@toc{#1}{#2}}}}% %% TODO - ok in plain?
1422 \def\babel@toc#1#2{%
1423   \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.

```
1424 \newif\ifbbl@usedategroup
1425 \def\bbl@switch#1{%  from select@, foreign@
1426   % make sure there is info for the language if so requested
1427   \bbl@ensureinfo{#1}%
1428   % restore
1429   \originalTeX
1430   \expandafter\def\expandafter\originalTeX\expandafter{%
1431     \csname noextras#1\endcsname
1432     \let\originalTeX\@empty
1433     \babel@beginsave}%
1434   \bbl@usehooks{afterreset}{}%
1435   \languageshorthands{none}%
1436   % set the locale id
1437   \bbl@id@assign
1438   % switch captions, date
1439   % No text is supposed to be added here, so we remove any
1440   % spurious spaces.
1441   \bbl@bsphack
1442     \ifcase\bbl@select@type
1443       \csname captions#1\endcsname\relax
1444       \csname date#1\endcsname\relax
1445     \else
1446       \bbl@xin@{,captions,}{,\bbl@select@opts,}%
1447       \ifin@
1448         \csname captions#1\endcsname\relax
1449       \fi
1450       \bbl@xin@{,date,}{,\bbl@select@opts,}%
1451       \ifin@  % if \foreign... within \<lang>date
1452         \csname date#1\endcsname\relax
1453       \fi
1454     \fi
1455   \bbl@esphack
1456   % switch extras
1457   \bbl@usehooks{beforeextras}{}%
1458   \csname extras#1\endcsname\relax
1459   \bbl@usehooks{afterextras}{}%
1460   %  > babel-ensure
1461   %  > babel-sh-<short>
1462   %  > babel-bidi
1463   %  > babel-fontspec
1464   % hyphenation - case mapping
1465   \ifcase\bbl@opt@hyphenmap\or
1466     \def\BabelLower##1##2{\lccode##1=##2\relax}%
1467     \ifnum\bbl@hymapsel>4\else
```

97

```
1468        \csname\languagename @bbl@hyphenmap\endcsname
1469      \fi
1470      \chardef\bbl@opt@hyphenmap\z@
1471    \else
1472      \ifnum\bbl@hymapsel>\bbl@opt@hyphenmap\else
1473        \csname\languagename @bbl@hyphenmap\endcsname
1474      \fi
1475    \fi
1476    \global\let\bbl@hymapsel\@cclv
1477    % hyphenation - select patterns
1478    \bbl@patterns{#1}%
1479    % hyphenation - allow stretching with babelnohyphens
1480    \ifnum\language=\l@babelnohyphens
1481      \babel@savevariable\emergencystretch
1482      \emergencystretch\maxdimen
1483      \babel@savevariable\hbadness
1484      \hbadness\@M
1485    \fi
1486    % hyphenation - mins
1487    \babel@savevariable\lefthyphenmin
1488    \babel@savevariable\righthyphenmin
1489    \expandafter\ifx\csname #1hyphenmins\endcsname\relax
1490      \set@hyphenmins\tw@\thr@@\relax
1491    \else
1492      \expandafter\expandafter\expandafter\set@hyphenmins
1493        \csname #1hyphenmins\endcsname\relax
1494    \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.

```
1495 \long\def\otherlanguage#1{%
1496   \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\thr@@\fi
1497   \csname selectlanguage \endcsname{#1}%
1498   \ignorespaces}
```

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

```
1499 \long\def\endotherlanguage{%
1500   \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.

```
1501 \expandafter\def\csname otherlanguage*\endcsname{%
1502   \@ifnextchar[\bbl@otherlanguage@s{\bbl@otherlanguage@s[]}}
1503 \def\bbl@otherlanguage@s[#1]#2{%
1504   \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
1505   \def\bbl@select@opts{#1}%
1506   \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".

```
1507 \expandafter\let\csname endotherlanguage*\endcsname\relax
```

\foreignlanguage The \foreignlanguage command is another substitute for the \selectlanguage command. This command takes two arguments, the first argument is the name of the language to use for typesetting the text specified in the second argument.

Unlike \selectlanguage this command doesn't switch *everything*, it only switches the hyphenation rules and the extra definitions for the language specified. It does this within a group and assumes the \extras⟨*lang*⟩ command doesn't make any \global changes. The coding is very similar to part of \selectlanguage.

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

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

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

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

```
1508 \providecommand\bbl@beforeforeign{}
1509 \edef\foreignlanguage{%
1510   \noexpand\protect
1511   \expandafter\noexpand\csname foreignlanguage \endcsname}
1512 \expandafter\def\csname foreignlanguage \endcsname{%
1513   \@ifstar\bbl@foreign@s\bbl@foreign@x}
1514 \providecommand\bbl@foreign@x[3][]{%
1515   \begingroup
1516     \def\bbl@select@opts{#1}%
1517     \let\BabelText\@firstofone
1518     \bbl@beforeforeign
1519     \foreign@language{#2}%
1520     \bbl@usehooks{foreign}{}%
1521     \BabelText{#3}% Now in horizontal mode!
1522   \endgroup}
1523 \def\bbl@foreign@s#1#2{% TODO - \shapemode, \@setpar, ?\@@par
1524   \begingroup
1525     {\par}%
1526     \let\BabelText\@firstofone
1527     \foreign@language{#1}%
1528     \bbl@usehooks{foreign*}{}%
1529     \bbl@dirparastext
1530     \BabelText{#2}% Still in vertical mode!
1531     {\par}%
1532   \endgroup}
```

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

```
1533 \def\foreign@language#1{%
1534   % set name
1535   \edef\languagename{#1}%
1536   \ifbbl@usedategroup
1537     \bbl@add\bbl@select@opts{,date,}%
```

```
1538      \bbl@usedategroupfalse
1539    \fi
1540    \bbl@fixname\languagename
1541    % TODO. name@map here?
1542    \bbl@provide@locale
1543    \bbl@iflanguage\languagename{%
1544      \expandafter\ifx\csname date\languagename\endcsname\relax
1545        \bbl@warning    % TODO - why a warning, not an error?
1546          {Unknown language `#1'. Either you have\\%
1547           misspelled its name, it has not been installed,\\%
1548           or you requested it in a previous run. Fix its name,\\%
1549           install it or just rerun the file, respectively. In\\%
1550           some cases, you may need to remove the aux file.\\%
1551           I'll proceed, but expect wrong results.\\%
1552           Reported}%
1553      \fi
1554    % set type
1555    \let\bbl@select@type\@ne
1556    \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.

```
1557 \let\bbl@hyphlist\@empty
1558 \let\bbl@hyphenation@\relax
1559 \let\bbl@pttnlist\@empty
1560 \let\bbl@patterns@\relax
1561 \let\bbl@hymapsel=\@cclv
1562 \def\bbl@patterns#1{%
1563   \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
1564       \csname l@#1\endcsname
1565       \edef\bbl@tempa{#1}%
1566     \else
1567       \csname l@#1:\f@encoding\endcsname
1568       \edef\bbl@tempa{#1:\f@encoding}%
1569     \fi
1570   \@expandtwoargs\bbl@usehooks{patterns}{{#1}{\bbl@tempa}}%
1571   % > luatex
1572   \@ifundefined{bbl@hyphenation@}{}{% Can be \relax!
1573     \begingroup
1574       \bbl@xin@{,\number\language,}{,\bbl@hyphlist}%
1575       \ifin@\else
1576         \@expandtwoargs\bbl@usehooks{hyphenation}{{#1}{\bbl@tempa}}%
1577         \hyphenation{%
1578           \bbl@hyphenation@
1579           \@ifundefined{bbl@hyphenation@#1}%
1580             \@empty
1581             {\space\csname bbl@hyphenation@#1\endcsname}}%
1582         \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
1583       \fi
1584     \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*.

```
1585 \def\hyphenrules#1{%
1586   \edef\bbl@tempf{#1}%
1587   \bbl@fixname\bbl@tempf
1588   \bbl@iflanguage\bbl@tempf{%
1589     \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
1590     \ifx\languageshorthands\@undefined\else
1591       \languageshorthands{none}%
1592     \fi
1593     \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax
1594       \set@hyphenmins\tw@\thr@@\relax
1595     \else
1596       \expandafter\expandafter\expandafter\set@hyphenmins
1597       \csname\bbl@tempf hyphenmins\endcsname\relax
1598     \fi}}
1599 \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.

```
1600 \def\providehyphenmins#1#2{%
1601   \expandafter\ifx\csname #1hyphenmins\endcsname\relax
1602     \@namedef{#1hyphenmins}{#2}%
1603   \fi}
```

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

```
1604 \def\set@hyphenmins#1#2{%
1605   \lefthyphenmin#1\relax
1606   \righthyphenmin#2\relax}
```

\ProvidesLanguage   The identification code for each file is something that was introduced in LaTeX 2ε. 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.

```
1607 \ifx\ProvidesFile\@undefined
1608   \def\ProvidesLanguage#1[#2 #3 #4]{%
1609     \wlog{Language: #1 #4 #3 <#2>}%
1610     }
1611 \else
1612   \def\ProvidesLanguage#1{%
1613     \begingroup
1614       \catcode`\ 10 %
1615       \@makeother\/%
1616       \@ifnextchar[%
1617         {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}}
1618   \def\@provideslanguage#1[#2]{%
1619     \wlog{Language: #1 #2}%
1620     \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
1621     \endgroup}
1622 \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.

```
1623 \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.

```
1624 \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':

```
1625 \providecommand\setlocale{%
1626   \bbl@error
1627     {Not yet available}%
1628     {Find an armchair, sit down and wait}}
1629 \let\uselocale\setlocale
1630 \let\locale\setlocale
1631 \let\selectlocale\setlocale
1632 \let\localename\setlocale
1633 \let\textlocale\setlocale
1634 \let\textlanguage\setlocale
1635 \let\languagetext\setlocale
```

9.2 Errors

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

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

When the format knows about \PackageError it must be LaTeX2_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.

```
1636 \edef\bbl@nulllanguage{\string\language=0}
1637 \ifx\PackageError\@undefined  % TODO. Move to Plain
1638   \def\bbl@error#1#2{%
1639     \begingroup
1640       \newlinechar=`\^^J
1641       \def\\{^^J(babel) }%
1642       \errhelp{#2}\errmessage{\\#1}%
1643     \endgroup}
1644   \def\bbl@warning#1{%
1645     \begingroup
1646       \newlinechar=`\^^J
1647       \def\\{^^J(babel) }%
1648       \message{\\#1}%
1649     \endgroup}
1650   \let\bbl@infowarn\bbl@warning
1651   \def\bbl@info#1{%
1652     \begingroup
1653       \newlinechar=`\^^J
1654       \def\\{^^J}%
1655       \wlog{#1}%
1656     \endgroup}
1657 \fi
1658 \def\bbl@nocaption{\protect\bbl@nocaption@i}
```

```
1659 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
1660   \global\@namedef{#2}{\textbf{?#1?}}%
1661   \@nameuse{#2}%
1662   \bbl@warning{%
1663     \@backslashchar#2 not set. Please, define it\\%
1664     after the language has been loaded (typically\\%
1665     in the preamble) with something like:\\%
1666     \string\renewcommand\@backslashchar#2{..}\\%
1667     Reported}}
1668 \def\bbl@tentative{\protect\bbl@tentative@i}
1669 \def\bbl@tentative@i#1{%
1670   \bbl@warning{%
1671     Some functions for '#1' are tentative.\\%
1672     They might not work as expected and their behavior\\%
1673     could change in the future.\\%
1674     Reported}}
1675 \def\@nolanerr#1{%
1676   \bbl@error
1677     {You haven't defined the language #1\space yet.\\%
1678      Perhaps you misspelled it or your installation\\%
1679      is not complete}%
1680     {Your command will be ignored, type <return> to proceed}}
1681 \def\@nopatterns#1{%
1682   \bbl@warning
1683     {No hyphenation patterns were preloaded for\\%
1684      the language `#1' into the format.\\%
1685      Please, configure your TeX system to add them and\\%
1686      rebuild the format. Now I will use the patterns\\%
1687      preloaded for \bbl@nulllanguage\space instead}}
1688 \let\bbl@usehooks\@gobbletwo
1689 \ifx\bbl@onlyswitch\@empty\endinput\fi
1690   % Here ended switch.def
```

 Here ended switch.def.

```
1691 \ifx\directlua\@undefined\else
1692   \ifx\bbl@luapatterns\@undefined
1693     \input luababel.def
1694   \fi
1695 \fi
1696 ⟨⟨Basic macros⟩⟩
1697 \bbl@trace{Compatibility with language.def}
1698 \ifx\bbl@languages\@undefined
1699   \ifx\directlua\@undefined
1700     \openin1 = language.def % TODO. Remove hardcoded number
1701     \ifeof1
1702       \closein1
1703       \message{I couldn't find the file language.def}
1704     \else
1705       \closein1
1706       \begingroup
1707         \def\addlanguage#1#2#3#4#5{%
1708           \expandafter\ifx\csname lang@#1\endcsname\relax\else
1709             \global\expandafter\let\csname l@#1\expandafter\endcsname
1710               \csname lang@#1\endcsname
1711           \fi}%
1712         \def\uselanguage#1{}%
1713         \input language.def
1714       \endgroup
1715     \fi
```

```
1716    \fi
1717    \chardef\l@english\z@
1718 \fi
```

\addto   It takes two arguments, a ⟨*control sequence*⟩ and T_EX-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.

```
1719 \def\addto#1#2{%
1720    \ifx#1\@undefined
1721      \def#1{#2}%
1722    \else
1723      \ifx#1\relax
1724        \def#1{#2}%
1725      \else
1726        {\toks@\expandafter{#1#2}%
1727          \xdef#1{\the\toks@}}%
1728      \fi
1729    \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?

```
1730 \def\bbl@withactive#1#2{%
1731    \begingroup
1732      \lccode`~=`#2\relax
1733      \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 L^AT_EX 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.

```
1734 \def\bbl@redefine#1{%
1735    \edef\bbl@tempa{\bbl@stripslash#1}%
1736    \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1737    \expandafter\def\csname\bbl@tempa\endcsname}
1738 \@onlypreamble\bbl@redefine
```

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

```
1739 \def\bbl@redefine@long#1{%
1740    \edef\bbl@tempa{\bbl@stripslash#1}%
1741    \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1742    \expandafter\long\expandafter\def\csname\bbl@tempa\endcsname}
1743 \@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␣.

```
1744 \def\bbl@redefinerobust#1{%
1745    \edef\bbl@tempa{\bbl@stripslash#1}%
```

```
1746    \bbl@ifunset{\bbl@tempa\space}%
1747      {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
1748       \bbl@exp{\def\\#1{\\\protect\<\bbl@tempa\space>}}}%
1749      {\bbl@exp{\let\<org@\bbl@tempa>\<\bbl@tempa\space>}}%
1750      \@namedef{\bbl@tempa\space}}
1751 \@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.

```
1752 \bbl@trace{Hooks}
1753 \newcommand\AddBabelHook[3][]{%
1754    \bbl@ifunset{bbl@hk@#2}{\EnableBabelHook{#2}}{}%
1755    \def\bbl@tempa##1,#3=##2,##3\@empty{\def\bbl@tempb{##2}}%
1756    \expandafter\bbl@tempa\bbl@evargs,#3=,\@empty
1757    \bbl@ifunset{bbl@ev@#2@#3@#1}%
1758      {\bbl@csarg\bbl@add{ev@#3@#1}{\bbl@elt{#2}}}%
1759      {\bbl@csarg\let{ev@#2@#3@#1}\relax}%
1760    \bbl@csarg\newcommand{ev@#2@#3@#1}[\bbl@tempb]}
1761 \newcommand\EnableBabelHook[1]{\bbl@csarg\let{hk@#1}\@firstofone}
1762 \newcommand\DisableBabelHook[1]{\bbl@csarg\let{hk@#1}\@gobble}
1763 \def\bbl@usehooks#1#2{%
1764    \def\bbl@elt##1{%
1765      \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#1@}#2}}%
1766    \bbl@cs{ev@#1@}%
1767    \ifx\languagename\@undefined\else % Test required for Plain (?)
1768      \def\bbl@elt##1{%
1769        \bbl@cs{hk@##1}{\bbl@cl{ev@##1@#1}#2}}%
1770      \bbl@cl{ev@#1}%
1771    \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).

```
1772 \def\bbl@evargs{,% <- don't delete this comma
1773    everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
1774    adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
1775    beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
1776    hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%
1777    beforestart=0,languagename=2}
```

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

```
1778 \bbl@trace{Defining babelensure}
1779 \newcommand\babelensure[2][]{%  TODO - revise test files
```

```
1780    \AddBabelHook{babel-ensure}{afterextras}{%
1781      \ifcase\bbl@select@type
1782        \bbl@cl{e}%
1783      \fi}%
1784    \begingroup
1785      \let\bbl@ens@include\@empty
1786      \let\bbl@ens@exclude\@empty
1787      \def\bbl@ens@fontenc{\relax}%
1788      \def\bbl@tempb##1{%
1789        \ifx\@empty##1\else\noexpand##1\expandafter\bbl@tempb\fi}%
1790      \edef\bbl@tempa{\bbl@tempb#1\@empty}%
1791      \def\bbl@tempb##1=##2\@@{\@namedef{bbl@ens@##1}{##2}}%
1792      \bbl@foreach\bbl@tempa{\bbl@tempb##1\@@}%
1793      \def\bbl@tempc{\bbl@ensure}%
1794      \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1795        \expandafter{\bbl@ens@include}}%
1796      \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1797        \expandafter{\bbl@ens@exclude}}%
1798      \toks@\expandafter{\bbl@tempc}%
1799      \bbl@exp{%
1800    \endgroup
1801    \def\<\bbl@e@#2>{\the\toks@{\bbl@ens@fontenc}}}}
1802  \def\bbl@ensure#1#2#3{% 1: include 2: exclude 3: fontenc
1803    \def\bbl@tempb##1{% elt for (excluding) \bbl@captionslist list
1804      \ifx##1\@undefined % 3.32 - Don't assume the macro exists
1805        \edef##1{\noexpand\bbl@nocaption
1806          {\bbl@stripslash##1}{\languagename\bbl@stripslash##1}}%
1807      \fi
1808      \ifx##1\@empty\else
1809        \in@{##1}{#2}%
1810        \ifin@\else
1811          \bbl@ifunset{bbl@ensure@\languagename}%
1812            {\bbl@exp{%
1813              \\\DeclareRobustCommand\<bbl@ensure@\languagename>[1]{%
1814                \\\foreignlanguage{\languagename}%
1815                {\ifx\relax#3\else
1816                  \\\fontencoding{#3}\\\selectfont
1817                  \fi
1818                  ########1}}}}%
1819            {}%
1820        \toks@\expandafter{##1}%
1821        \edef##1{%
1822          \bbl@csarg\noexpand{ensure@\languagename}%
1823          {\the\toks@}}%
1824      \fi
1825      \expandafter\bbl@tempb
1826    \fi}%
1827  \expandafter\bbl@tempb\bbl@captionslist\today\@empty
1828  \def\bbl@tempa##1{% elt for include list
1829    \ifx##1\@empty\else
1830      \bbl@csarg\in@{ensure@\languagename\expandafter}\expandafter{##1}%
1831      \ifin@\else
1832        \bbl@tempb##1\@empty
1833      \fi
1834      \expandafter\bbl@tempa
1835    \fi}%
1836  \bbl@tempa#1\@empty}
1837  \def\bbl@captionslist{%
1838    \prefacename\refname\abstractname\bibname\chaptername\appendixname
```

```
1839    \contentsname\listfigurename\listtablename\indexname\figurename
1840    \tablename\partname\enclname\ccname\headtoname\pagename\seename
1841    \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.

```
1842 \bbl@trace{Macros for setting language files up}
1843 \def\bbl@ldfinit{% TODO. Merge into the next macro? Unused elsewhere
1844    \let\bbl@screset\@empty
1845    \let\BabelStrings\bbl@opt@string
1846    \let\BabelOptions\@empty
1847    \let\BabelLanguages\relax
1848    \ifx\originalTeX\@undefined
1849      \let\originalTeX\@empty
1850    \else
1851      \originalTeX
1852    \fi}
1853 \def\LdfInit#1#2{%
1854    \chardef\atcatcode=\catcode`\@
1855    \catcode`\@=11\relax
1856    \chardef\eqcatcode=\catcode`\=
1857    \catcode`\==12\relax
1858    \expandafter\if\expandafter\@backslashchar
1859                  \expandafter\@car\string#2\@nil
1860      \ifx#2\@undefined\else
1861        \ldf@quit{#1}%
1862      \fi
1863    \else
1864      \expandafter\ifx\csname#2\endcsname\relax\else
1865        \ldf@quit{#1}%
1866      \fi
1867    \fi
1868    \bbl@ldfinit}
```

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

```
1869 \def\ldf@quit#1{%
1870    \expandafter\main@language\expandafter{#1}%
```

```
1871    \catcode`\@=\atcatcode \let\atcatcode\relax
1872    \catcode`\==\eqcatcode \let\eqcatcode\relax
1873    \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.

```
1874 \def\bbl@afterldf#1{% TODO. Merge into the next macro? Unused elsewhere
1875    \bbl@afterlang
1876    \let\bbl@afterlang\relax
1877    \let\BabelModifiers\relax
1878    \let\bbl@screset\relax}%
1879 \def\ldf@finish#1{%
1880    \ifx\loadlocalcfg\@undefined\else % For LaTeX 209
1881      \loadlocalcfg{#1}%
1882    \fi
1883    \bbl@afterldf{#1}%
1884    \expandafter\main@language\expandafter{#1}%
1885    \catcode`\@=\atcatcode \let\atcatcode\relax
1886    \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.

```
1887 \@onlypreamble\LdfInit
1888 \@onlypreamble\ldf@quit
1889 \@onlypreamble\ldf@finish
```

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

```
1890 \def\main@language#1{%
1891    \def\bbl@main@language{#1}%
1892    \let\languagename\bbl@main@language % TODO. Set localename
1893    \bbl@id@assign
1894    \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.

```
1895 \def\bbl@beforestart{%
1896    \bbl@usehooks{beforestart}{}%
1897    \global\let\bbl@beforestart\relax}
1898 \AtBeginDocument{%
1899    \@nameuse{bbl@beforestart}%
1900    \if@filesw
1901      \providecommand\babel@aux[2]{}%
1902      \immediate\write\@mainaux{%
1903        \string\providecommand\string\babel@aux[2]{}}%
1904      \immediate\write\@mainaux{\string\@nameuse{bbl@beforestart}}%
1905    \fi
1906    \expandafter\selectlanguage\expandafter{\bbl@main@language}%
1907    \ifbbl@single  % must go after the line above.
1908      \renewcommand\selectlanguage[1]{}%
1909      \renewcommand\foreignlanguage[2]{#2}%
```

```
1910      \global\let\babel@aux\@gobbletwo  % Also as flag
1911    \fi
1912    \ifcase\bbl@engine\or\pagedir\bodydir\fi} % TODO - a better place
```

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

```
1913 \def\select@language@x#1{%
1914   \ifcase\bbl@select@type
1915     \bbl@ifsamestring\languagename{#1}{}{\select@language{#1}}%
1916   \else
1917     \select@language{#1}%
1918   \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.

```
1919 \bbl@trace{Shorhands}
1920 \def\bbl@add@special#1{% 1:a macro like \", \?, etc.
1921   \bbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
1922   \bbl@ifunset{@sanitize}{}{\bbl@add\@sanitize{\@makeother#1}}%
1923   \ifx\nfss@catcodes\@undefined\else % TODO - same for above
1924     \begingroup
1925       \catcode`#1\active
1926       \nfss@catcodes
1927       \ifnum\catcode`#1=\active
1928         \endgroup
1929         \bbl@add\nfss@catcodes{\@makeother#1}%
1930       \else
1931         \endgroup
1932       \fi
1933   \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.

```
1934 \def\bbl@remove@special#1{%
1935   \begingroup
1936     \def\x##1##2{\ifnum`#1=`##2\noexpand\@empty
1937                 \else\noexpand##1\noexpand##2\fi}%
1938     \def\do{\x\do}%
1939     \def\@makeother{\x\@makeother}%
1940   \edef\x{\endgroup
1941     \def\noexpand\dospecials{\dospecials}%
1942     \expandafter\ifx\csname @sanitize\endcsname\relax\else
1943       \def\noexpand\@sanitize{\@sanitize}%
1944     \fi}%
1945   \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

109

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).

```
1946 \def\bbl@active@def#1#2#3#4{%
1947   \@namedef{#3#1}{%
1948     \expandafter\ifx\csname#2@sh@#1@\endcsname\relax
1949       \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
1950     \else
1951       \bbl@afterfi\csname#2@sh@#1@\endcsname
1952     \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.

```
1953   \long\@namedef{#3@arg#1}##1{%
1954     \expandafter\ifx\csname#2@sh@#1@\string##1@\endcsname\relax
1955       \bbl@afterelse\csname#4#1\endcsname##1%
1956     \else
1957       \bbl@afterfi\csname#2@sh@#1@\string##1@\endcsname
1958     \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.

```
1959 \def\initiate@active@char#1{%
1960   \bbl@ifunset{active@char\string#1}%
1961     {\bbl@withactive
1962       {\expandafter\@initiate@active@char\expandafter}#1\string#1#1}%
1963     {}}
```

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`).

```
1964 \def\@initiate@active@char#1#2#3{%
1965   \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
1966   \ifx#1\@undefined
1967     \bbl@csarg\edef{oridef@#2}{\let\noexpand#1\noexpand\@undefined}%
1968   \else
1969     \bbl@csarg\let{oridef@@#2}#1%
1970     \bbl@csarg\edef{oridef@#2}{%
1971       \let\noexpand#1%
1972       \expandafter\noexpand\csname bbl@oridef@@#2\endcsname}%
1973   \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

110

\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*).

```
1974  \ifx#1#3\relax
1975    \expandafter\let\csname normal@char#2\endcsname#3%
1976  \else
1977    \bbl@info{Making #2 an active character}%
1978    \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
1979      \@namedef{normal@char#2}{%
1980        \textormath{#3}{\csname bbl@oridef@@#2\endcsname}}%
1981    \else
1982      \@namedef{normal@char#2}{#3}%
1983    \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).

```
1984    \bbl@restoreactive{#2}%
1985    \AtBeginDocument{%
1986      \catcode`#2\active
1987      \if@filesw
1988        \immediate\write\@mainaux{\catcode`\string#2\active}%
1989      \fi}%
1990    \expandafter\bbl@add@special\csname#2\endcsname
1991    \catcode`#2\active
1992  \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*⟩).

```
1993    \let\bbl@tempa\@firstoftwo
1994    \if\string^#2%
1995      \def\bbl@tempa{\noexpand\textormath}%
1996    \else
1997      \ifx\bbl@mathnormal\@undefined\else
1998        \let\bbl@tempa\bbl@mathnormal
1999      \fi
2000    \fi
2001    \expandafter\edef\csname active@char#2\endcsname{%
2002      \bbl@tempa
2003        {\noexpand\if@safe@actives
2004          \noexpand\expandafter
2005          \expandafter\noexpand\csname normal@char#2\endcsname
2006        \noexpand\else
2007          \noexpand\expandafter
2008          \expandafter\noexpand\csname bbl@doactive#2\endcsname
2009        \noexpand\fi}%
2010      {\expandafter\noexpand\csname normal@char#2\endcsname}}%
2011    \bbl@csarg\edef{doactive#2}{%
2012      \expandafter\noexpand\csname user@active#2\endcsname}%
```

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

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

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

```
2013   \bbl@csarg\edef{active@#2}{%
2014     \noexpand\active@prefix\noexpand#1%
2015     \expandafter\noexpand\csname active@char#2\endcsname}%
2016   \bbl@csarg\edef{normal@#2}{%
2017     \noexpand\active@prefix\noexpand#1%
2018     \expandafter\noexpand\csname normal@char#2\endcsname}%
2019   \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.

```
2020   \bbl@active@def#2\user@group{user@active}{language@active}%
2021   \bbl@active@def#2\language@group{language@active}{system@active}%
2022   \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.

```
2023   \expandafter\edef\csname\user@group @sh@#2@@\endcsname
2024     {\expandafter\noexpand\csname normal@char#2\endcsname}%
2025   \expandafter\edef\csname\user@group @sh@#2@\string\protect@\endcsname
2026     {\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.

```
2027   \if\string'#2%
2028     \let\prim@s\bbl@prim@s
2029     \let\active@math@prime#1%
2030   \fi
2031   \bbl@usehooks{initiateactive}{{#1}{#2}{#3}}}
```

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

```
2032 ⟨⟨*More package options⟩⟩ ≡
2033 \DeclareOption{math=active}{}
2034 \DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
2035 ⟨⟨/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.

```
2036 \@ifpackagewith{babel}{KeepShorthandsActive}%
2037   {\let\bbl@restoreactive\@gobble}%
2038   {\def\bbl@restoreactive#1{%
2039     \bbl@exp{%
2040       \\\AfterBabelLanguage\\\CurrentOption
2041         {\catcode`#1=\the\catcode`#1\relax}%
2042       \\\AtEndOfPackage
```

112

```
2043          {\catcode`#1=\the\catcode`#1\relax}}}%
2044     \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.

```
2045 \def\bbl@sh@select#1#2{%
2046    \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
2047      \bbl@afterelse\bbl@scndcs
2048    \else
2049      \bbl@afterfi\csname#1@sh@#2@sel\endcsname
2050    \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.

```
2051 \begingroup
2052 \bbl@ifunset{ifincsname}% TODO. Ugly. Correct?
2053    {\gdef\active@prefix#1{%
2054       \ifx\protect\@typeset@protect
2055       \else
2056         \ifx\protect\@unexpandable@protect
2057           \noexpand#1%
2058         \else
2059           \protect#1%
2060         \fi
2061         \expandafter\@gobble
2062       \fi}}
2063    {\gdef\active@prefix#1{%
2064       \ifincsname
2065         \string#1%
2066         \expandafter\@gobble
2067       \else
2068         \ifx\protect\@typeset@protect
2069         \else
2070           \ifx\protect\@unexpandable@protect
2071             \noexpand#1%
2072           \else
2073             \protect#1%
2074           \fi
2075           \expandafter\expandafter\expandafter\@gobble
2076         \fi
2077      \fi}}
2078 \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*⟩.

```
2079 \newif\if@safe@actives
2080 \@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.

```
2081 \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`.

```
2082 \def\bbl@activate#1{%
2083   \bbl@withactive{\expandafter\let\expandafter}#1%
2084     \csname bbl@active@\string#1\endcsname}
2085 \def\bbl@deactivate#1{%
2086   \bbl@withactive{\expandafter\let\expandafter}#1%
2087     \csname bbl@normal@\string#1\endcsname}
```

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

```
2088 \def\bbl@firstcs#1#2{\csname#1\endcsname}
2089 \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.

```
2090 \def\bbl@texormathorpdf#1#2#3{%
2091   \ifx\texorpdfstring\@undefined
2092     \textormath{#1}{#2}%
2093   \else
2094     \texorpdfstring{\textormath{#1}{#2}}{#3}%
2095   \fi}
2096 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
2097 \def\@decl@short#1#2#3\@nil#4{%
2098   \def\bbl@tempa{#3}%
2099   \ifx\bbl@tempa\@empty
2100     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
2101     \bbl@ifunset{#1@sh@\string#2@}{}%
2102       {\def\bbl@tempa{#4}%
2103        \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbl@tempa
2104        \else
2105          \bbl@info
2106            {Redefining #1 shorthand \string#2\\%
2107             in language \CurrentOption}%
2108        \fi}%
2109     \@namedef{#1@sh@\string#2@}{#4}%
2110   \else
2111     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs
2112     \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%
2113       {\def\bbl@tempa{#4}%
2114        \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
2115        \else
2116          \bbl@info
2117            {Redefining #1 shorthand \string#2\string#3\\%
2118             in language \CurrentOption}%
2119        \fi}%
2120     \@namedef{#1@sh@\string#2@\string#3@}{#4}%
2121   \fi}
```

114

\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.

```
2122 \def\textormath{%
2123   \ifmmode
2124     \expandafter\@secondoftwo
2125   \else
2126     \expandafter\@firstoftwo
2127   \fi}
```

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

```
2128 \def\user@group{user}
2129 \def\language@group{english} % TODO. I don't like defaults
2130 \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.

```
2131 \def\useshorthands{%
2132   \@ifstar\bbl@usesh@s{\bbl@usesh@x{}}}
2133 \def\bbl@usesh@s#1{%
2134   \bbl@usesh@x
2135     {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bbl@activate{#1}}}%
2136     {#1}}
2137 \def\bbl@usesh@x#1#2{%
2138   \bbl@ifshorthand{#2}%
2139     {\def\user@group{user}%
2140      \initiate@active@char{#2}%
2141      #1%
2142      \bbl@activate{#2}}%
2143     {\bbl@error
2144        {Cannot declare a shorthand turned off (\string#2)}
2145        {Sorry, but you cannot use shorthands which have been\\%
2146         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.

```
2147 \def\user@language@group{user@\language@group}
2148 \def\bbl@set@user@generic#1#2{%
2149   \bbl@ifunset{user@generic@active#1}%
2150     {\bbl@active@def#1\user@language@group{user@active}{user@generic@active}%
2151      \bbl@active@def#1\user@group{user@generic@active}{language@active}%
2152      \expandafter\edef\csname#2@sh@#1@@\endcsname{%
2153        \expandafter\noexpand\csname normal@char#1\endcsname}%
2154      \expandafter\edef\csname#2@sh@#1@\string\protect@\endcsname{%
2155        \expandafter\noexpand\csname user@active#1\endcsname}}%
2156   \@empty}
2157 \newcommand\defineshorthand[3][user]{%
2158   \edef\bbl@tempa{\zap@space#1 \@empty}%
2159   \bbl@for\bbl@tempb\bbl@tempa{%
2160     \if*\expandafter\@car\bbl@tempb\@nil
```

115

```
2161        \edef\bbl@tempb{user@\expandafter\@gobble\bbl@tempb}%
2162        \@expandtwoargs
2163          \bbl@set@user@generic{\expandafter\string\@car#2\@nil}\bbl@tempb
2164      \fi
2165      \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].

```
2166 \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".

```
2167 \def\aliasshorthand#1#2{%
2168   \bbl@ifshorthand{#2}%
2169     {\expandafter\ifx\csname active@char\string#2\endcsname\relax
2170        \ifx\document\@notprerr
2171          \@notshorthand{#2}%
2172        \else
2173          \initiate@active@char{#2}%
2174          \expandafter\let\csname active@char\string#2\expandafter\endcsname
2175            \csname active@char\string#1\endcsname
2176          \expandafter\let\csname normal@char\string#2\expandafter\endcsname
2177            \csname normal@char\string#1\endcsname
2178          \bbl@activate{#2}%
2179        \fi
2180      \fi}%
2181     {\bbl@error
2182       {Cannot declare a shorthand turned off (\string#2)}
2183       {Sorry, but you cannot use shorthands which have been\\%
2184        turned off in the package options}}}
```

\@notshorthand

```
2185 \def\@notshorthand#1{%
2186   \bbl@error{%
2187     The character `\string #1' should be made a shorthand character;\\%
2188     add the command \string\useshorthands\string{#1\string} to
2189     the preamble.\\%
2190     I will ignore your instruction}%
2191     {You may proceed, but expect unexpected results}}
```

\shorthandon The first level definition of these macros just passes the argument on to \bbl@switch@sh,
\shorthandoff adding \@nil at the end to denote the end of the list of characters.

```
2192 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
2193 \DeclareRobustCommand*\shorthandoff{%
2194   \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
2195 \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.

116

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.

```
2196 \def\bbl@switch@sh#1#2{%
2197   \ifx#2\@nnil\else
2198     \bbl@ifunset{bbl@active@\string#2}%
2199       {\bbl@error
2200         {I cannot switch `\string#2' on or off--not a shorthand}%
2201         {This character is not a shorthand. Maybe you made\\%
2202          a typing mistake? I will ignore your instruction}}%
2203       {\ifcase#1%
2204         \catcode`#212\relax
2205        \or
2206         \catcode`#2\active
2207        \or
2208         \csname bbl@oricat@\string#2\endcsname
2209         \csname bbl@oridef@\string#2\endcsname
2210       \fi}%
2211     \bbl@afterfi\bbl@switch@sh#1%
2212   \fi}
```

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

```
2213 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
2214 \def\bbl@putsh#1{%
2215   \bbl@ifunset{bbl@active@\string#1}%
2216     {\bbl@putsh@i#1\@empty\@nnil}%
2217     {\csname bbl@active@\string#1\endcsname}}
2218 \def\bbl@putsh@i#1#2\@nnil{%
2219   \csname\language@group @sh@\string#1@%
2220     \ifx\@empty#2\else\string#2@\fi\endcsname}
2221 \ifx\bbl@opt@shorthands\@nnil\else
2222   \let\bbl@s@initiate@active@char\initiate@active@char
2223   \def\initiate@active@char#1{%
2224     \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
2225   \let\bbl@s@switch@sh\bbl@switch@sh
2226   \def\bbl@switch@sh#1#2{%
2227     \ifx#2\@nnil\else
2228       \bbl@afterfi
2229       \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
2230     \fi}
2231   \let\bbl@s@activate\bbl@activate
2232   \def\bbl@activate#1{%
2233     \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
2234   \let\bbl@s@deactivate\bbl@deactivate
2235   \def\bbl@deactivate#1{%
2236     \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
2237 \fi
```

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

```
2238 \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.

117

```
2239 \def\bbl@prim@s{%
2240   \prime\futurelet\@let@token\bbl@pr@m@s}
2241 \def\bbl@if@primes#1#2{%
2242   \ifx#1\@let@token
2243     \expandafter\@firstoftwo
2244   \else\ifx#2\@let@token
2245     \bbl@afterelse\expandafter\@firstoftwo
2246   \else
2247     \bbl@afterfi\expandafter\@secondoftwo
2248   \fi\fi}
2249 \begingroup
2250   \catcode`\^=7  \catcode`\*=\active  \lccode`\*=`\^
2251   \catcode`\'=12 \catcode`\"=\active  \lccode`\"=`\'
2252   \lowercase{%
2253     \gdef\bbl@pr@m@s{%
2254       \bbl@if@primes"'%
2255         \pr@@@s
2256         {\bbl@if@primes*^\pr@@@t\egroup}}}
2257 \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).

```
2258 \initiate@active@char{~}
2259 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
2260 \bbl@activate{~}
```

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

```
2261 \expandafter\def\csname OT1dqpos\endcsname{127}
2262 \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

```
2263 \ifx\f@encoding\@undefined
2264   \def\f@encoding{OT1}
2265 \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.

```
2266 \bbl@trace{Language attributes}
2267 \newcommand\languageattribute[2]{%
2268   \def\bbl@tempc{#1}%
2269   \bbl@fixname\bbl@tempc
2270   \bbl@iflanguage\bbl@tempc{%
2271     \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.

```
2272        \ifx\bbl@known@attribs\@undefined
2273          \in@false
2274        \else
2275          \bbl@xin@{,\bbl@tempc-##1,}{,\bbl@known@attribs,}%
2276        \fi
2277        \ifin@
2278          \bbl@warning{%
2279            You have more than once selected the attribute '##1'\\%
2280            for language #1. Reported}%
2281        \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.

```
2282        \bbl@exp{%
2283          \\\bbl@add@list\\\bbl@known@attribs{\bbl@tempc-##1}}%
2284        \edef\bbl@tempa{\bbl@tempc-##1}%
2285        \expandafter\bbl@ifknown@ttrib\expandafter{\bbl@tempa}\bbl@attributes%
2286        {\csname\bbl@tempc @attr@##1\endcsname}%
2287        {\@attrerr{\bbl@tempc}{##1}}%
2288      \fi}}}
2289 \@onlypreamble\languageattribute
```

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

```
2290 \newcommand*{\@attrerr}[2]{%
2291   \bbl@error
2292     {The attribute #2 is unknown for language #1.}%
2293     {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}.

```
2294 \def\bbl@declare@ttribute#1#2#3{%
2295   \bbl@xin@{,#2,}{,\BabelModifiers,}%
2296   \ifin@
2297     \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
2298   \fi
2299   \bbl@add@list\bbl@attributes{#1-#2}%
2300   \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.
First we need to find out if any attributes were set; if not we're done. Then we need to check the list of known attributes. When we're this far \ifin@ has a value indicating if the attribute in question was set or not. Just to be safe the code to be executed is 'thrown over the \fi'.

```
2301 \def\bbl@ifattributeset#1#2#3#4{%
```

119

```
2302    \ifx\bbl@known@attribs\@undefined
2303      \in@false
2304    \else
2305      \bbl@xin@{,#1-#2,}{,\bbl@known@attribs,}%
2306    \fi
2307    \ifin@
2308      \bbl@afterelse#3%
2309    \else
2310      \bbl@afterfi#4%
2311    \fi
2312    }
```

\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. When a match is found the definition of \bbl@tempa is changed.
Finally we execute \bbl@tempa.

```
2313 \def\bbl@ifknown@ttrib#1#2{%
2314    \let\bbl@tempa\@secondoftwo
2315    \bbl@loopx\bbl@tempb{#2}{%
2316      \expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{,#1,}%
2317      \ifin@
2318        \let\bbl@tempa\@firstoftwo
2319      \else
2320      \fi}%
2321    \bbl@tempa
2322 }
```

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

```
2323 \def\bbl@clear@ttribs{%
2324    \ifx\bbl@attributes\@undefined\else
2325      \bbl@loopx\bbl@tempa{\bbl@attributes}{%
2326        \expandafter\bbl@clear@ttrib\bbl@tempa.
2327        }%
2328      \let\bbl@attributes\@undefined
2329    \fi}
2330 \def\bbl@clear@ttrib#1-#2.{%
2331    \expandafter\let\csname#1@attr@#2\endcsname\@undefined}
2332 \AtBeginDocument{\bbl@clear@ttribs}
```

9.7 Support for saving macro definitions

To save the meaning of control sequences using \babel@save, we use temporary control
sequences. To save hash table entries for these control sequences, we don't use the name
of the control sequence to be saved to construct the temporary name. Instead we simply
use the value of a counter, which is reset to zero each time we begin to save new values.
This works well because we release the saved meanings before we begin to save a new set
of control sequence meanings (see \selectlanguage and \originalTeX). Note undefined
macros are not undefined any more when saved – they are \relax'ed.

\babel@savecnt   The initialization of a new save cycle: reset the counter to zero.
\babel@beginsave
```
2333 \bbl@trace{Macros for saving definitions}
2334 \def\babel@beginsave{\babel@savecnt\z@}
```

120

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

```
2335 \newcount\babel@savecnt
2336 \babel@beginsave
```

\babel@save
\babel@savevariable

The macro \babel@save⟨*csname*⟩ saves the current meaning of the control sequence ⟨*csname*⟩ to \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.

```
2337 \def\babel@save#1{%
2338   \expandafter\let\csname babel@\number\babel@savecnt\endcsname#1\relax
2339   \toks@\expandafter{\originalTeX\let#1=}%
2340   \bbl@exp{%
2341     \def\\\originalTeX{\the\toks@\<babel@\number\babel@savecnt>\relax}}%
2342   \advance\babel@savecnt\@ne}
2343 \def\babel@savevariable#1{%
2344   \toks@\expandafter{\originalTeX #1=}%
2345   \bbl@exp{\def\\\originalTeX{\the\toks@\the#1\relax}}}
```

\bbl@frenchspacing
\bbl@nonfrenchspacing

Some languages need to have \frenchspacing in effect. Others don't want that. The command \bbl@frenchspacing switches it on when it isn't already in effect and \bbl@nonfrenchspacing switches it off if necessary.

```
2346 \def\bbl@frenchspacing{%
2347   \ifnum\the\sfcode`\.=\@m
2348     \let\bbl@nonfrenchspacing\relax
2349   \else
2350     \frenchspacing
2351     \let\bbl@nonfrenchspacing\nonfrenchspacing
2352   \fi}
2353 \let\bbl@nonfrenchspacing\nonfrenchspacing
2354 %
2355 \let\bbl@elt\relax
2356 \edef\bbl@fs@chars{%
2357   \bbl@elt{\string.}\@m{3000}\bbl@elt{\string?}\@m{3000}%
2358   \bbl@elt{\string!}\@m{3000}\bbl@elt{\string:}\@m{2000}%
2359   \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.

```
2360 \bbl@trace{Short tags}
2361 \def\babeltags#1{%
2362   \edef\bbl@tempa{\zap@space#1 \@empty}%
2363   \def\bbl@tempb##1=##2\@@{%
2364     \edef\bbl@tempc{%
2365       \noexpand\newcommand
2366       \expandafter\noexpand\csname ##1\endcsname{%
2367         \noexpand\protect
2368         \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}
2369       \noexpand\newcommand
2370       \expandafter\noexpand\csname text##1\endcsname{%
2371         \noexpand\foreignlanguage{##2}}}%
2372     \bbl@tempc}%
```

---

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

```
2373    \bbl@for\bbl@tempa\bbl@tempa{%
2374       \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.

```
2375 \bbl@trace{Hyphens}
2376 \@onlypreamble\babelhyphenation
2377 \AtEndOfPackage{%
2378   \newcommand\babelhyphenation[2][\@empty]{%
2379     \ifx\bbl@hyphenation@\relax
2380       \let\bbl@hyphenation@\@empty
2381     \fi
2382     \ifx\bbl@hyphlist\@empty\else
2383       \bbl@warning{%
2384         You must not intermingle \string\selectlanguage\space and\\%
2385         \string\babelhyphenation\space or some exceptions will not\\%
2386         be taken into account. Reported}%
2387     \fi
2388     \ifx\@empty#1%
2389       \protected@edef\bbl@hyphenation@{\bbl@hyphenation@\space#2}%
2390     \else
2391       \bbl@vforeach{#1}{%
2392         \def\bbl@tempa{##1}%
2393         \bbl@fixname\bbl@tempa
2394         \bbl@iflanguage\bbl@tempa{%
2395           \bbl@csarg\protected@edef{hyphenation@\bbl@tempa}{%
2396             \bbl@ifunset{bbl@hyphenation@\bbl@tempa}%
2397               \@empty
2398               {\csname bbl@hyphenation@\bbl@tempa\endcsname\space}%
2399             #2}}}%
2400     \fi}}
```

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

```
2401 \def\bbl@allowhyphens{\ifvmode\else\nobreak\hskip\z@skip\fi}
2402 \def\bbl@t@one{T1}
2403 \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.

```
2404 \newcommand\babelnullhyphen{\char\hyphenchar\font}
2405 \def\babelhyphen{\active@prefix\babelhyphen\bbl@hyphen}
2406 \def\bbl@hyphen{%
2407   \@ifstar{\bbl@hyphen@i @}{\bbl@hyphen@i\@empty}}
2408 \def\bbl@hyphen@i#1#2{%
2409   \bbl@ifunset{bbl@hy@#1#2\@empty}%
2410     {\csname bbl@#1usehyphen\endcsname{\discretionary{#2}{}{#2}}}%
2411     {\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,

---

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

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.

```
2412 \def\bbl@usehyphen#1{%
2413   \leavevmode
2414   \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
2415   \nobreak\hskip\z@skip}
2416 \def\bbl@@usehyphen#1{%
2417   \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}
```

The following macro inserts the hyphen char.

```
2418 \def\bbl@hyphenchar{%
2419   \ifnum\hyphenchar\font=\m@ne
2420     \babelnullhyphen
2421   \else
2422     \char\hyphenchar\font
2423   \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.

```
2424 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}{}}}
2425 \def\bbl@hy@@soft{\bbl@@usehyphen{\discretionary{\bbl@hyphenchar}{}{}}}
2426 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
2427 \def\bbl@hy@@hard{\bbl@@usehyphen\bbl@hyphenchar}
2428 \def\bbl@hy@nobreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}}
2429 \def\bbl@hy@@nobreak{\mbox{\bbl@hyphenchar}}
2430 \def\bbl@hy@repeat{%
2431   \bbl@usehyphen{%
2432     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
2433 \def\bbl@hy@@repeat{%
2434   \bbl@@usehyphen{%
2435     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
2436 \def\bbl@hy@empty{\hskip\z@skip}
2437 \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.

```
2438 \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.

```
2439 \bbl@trace{Multiencoding strings}
2440 \def\bbl@toglobal#1{\global\let#1#1}
2441 \def\bbl@recatcode#1{% TODO. Used only once?
2442   \@tempcnta="7F
2443   \def\bbl@tempa{%
2444     \ifnum\@tempcnta>"FF\else
2445       \catcode\@tempcnta=#1\relax
```

```
2446        \advance\@tempcnta\@ne
2447        \expandafter\bbl@tempa
2448      \fi}%
2449    \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).

```
2450 \@ifpackagewith{babel}{nocase}%
2451   {\let\bbl@patchuclc\relax}%
2452   {\def\bbl@patchuclc{%
2453     \global\let\bbl@patchuclc\relax
2454     \g@addto@macro\@uclclist{\reserved@b{\reserved@b\bbl@uclc}}%
2455     \gdef\bbl@uclc##1{%
2456       \let\bbl@encoded\bbl@encoded@uclc
2457       \bbl@ifunset{\languagename @bbl@uclc}% and resumes it
2458         {##1}%
2459         {\let\bbl@tempa##1\relax % Used by LANG@bbl@uclc
2460          \csname\languagename @bbl@uclc\endcsname}%
2461       {\bbl@tolower\@empty}{\bbl@toupper\@empty}}%
2462     \gdef\bbl@tolower{\csname\languagename @bbl@lc\endcsname}%
2463     \gdef\bbl@toupper{\csname\languagename @bbl@uc\endcsname}}}
```

```
2464 ⟨⟨*More package options⟩⟩ ≡
2465 \DeclareOption{nocase}{}
2466 ⟨⟨/More package options⟩⟩
```

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

```
2467 ⟨⟨*More package options⟩⟩ ≡
2468 \let\bbl@opt@strings\@nnil % accept strings=value
2469 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
2470 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
2471 \def\BabelStringsDefault{generic}
2472 ⟨⟨/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.

```
2473 \@onlypreamble\StartBabelCommands
2474 \def\StartBabelCommands{%
2475   \begingroup
2476   \bbl@recatcode{11}%
2477   ⟨⟨Macros local to BabelCommands⟩⟩
2478   \def\bbl@provstring##1##2{%
2479     \providecommand##1{##2}%
2480     \bbl@toglobal##1}%
2481   \global\let\bbl@scafter\@empty
2482   \let\StartBabelCommands\bbl@startcmds
2483   \ifx\BabelLanguages\relax
```

124

```
2484        \let\BabelLanguages\CurrentOption
2485      \fi
2486      \begingroup
2487      \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
2488      \StartBabelCommands}
2489 \def\bbl@startcmds{%
2490      \ifx\bbl@screset\@nnil\else
2491        \bbl@usehooks{stopcommands}{}%
2492      \fi
2493      \endgroup
2494      \begingroup
2495      \@ifstar
2496        {\ifx\bbl@opt@strings\@nnil
2497            \let\bbl@opt@strings\BabelStringsDefault
2498         \fi
2499         \bbl@startcmds@i}%
2500        \bbl@startcmds@i}
2501 \def\bbl@startcmds@i#1#2{%
2502      \edef\bbl@L{\zap@space#1 \@empty}%
2503      \edef\bbl@G{\zap@space#2 \@empty}%
2504      \bbl@startcmds@ii}
2505 \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.

```
2506 \newcommand\bbl@startcmds@ii[1][\@empty]{%
2507      \let\SetString\@gobbletwo
2508      \let\bbl@stringdef\@gobbletwo
2509      \let\AfterBabelCommands\@gobble
2510      \ifx\@empty#1%
2511        \def\bbl@sc@label{generic}%
2512        \def\bbl@encstring##1##2{%
2513          \ProvideTextCommandDefault##1{##2}%
2514          \bbl@toglobal##1%
2515          \expandafter\bbl@toglobal\csname\string?\string##1\endcsname}%
2516        \let\bbl@sctest\in@true
2517      \else
2518        \let\bbl@sc@charset\space % <- zapped below
2519        \let\bbl@sc@fontenc\space % <-    "        "
2520        \def\bbl@tempa##1=##2\@nil{%
2521          \bbl@csarg\edef{sc@\zap@space##1 \@empty}{##2 }}%
2522        \bbl@vforeach{label=#1}{\bbl@tempa##1\@nil}%
2523        \def\bbl@tempa##1 ##2{% space -> comma
2524          ##1%
2525          \ifx\@empty##2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa##2\fi}%
2526        \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%
2527        \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%
2528        \edef\bbl@sc@charset{\expandafter\zap@space\bbl@sc@charset\@empty}%
2529        \def\bbl@encstring##1##2{%
2530          \bbl@foreach\bbl@sc@fontenc{%
```

```
2531        \bbl@ifunset{T@####1}%
2532          {}%
2533          {\ProvideTextCommand##1{####1}{##2}%
2534           \bbl@toglobal##1%
2535           \expandafter
2536           \bbl@toglobal\csname####1\string##1\endcsname}}}%
2537      \def\bbl@sctest{%
2538        \bbl@xin@{,\bbl@opt@strings,}{,\bbl@sc@label,\bbl@sc@fontenc,}}%
2539    \fi
2540    \ifx\bbl@opt@strings\@nnil        % ie, no strings key -> defaults
2541    \else\ifx\bbl@opt@strings\relax   % ie, strings=encoded
2542      \let\AfterBabelCommands\bbl@aftercmds
2543      \let\SetString\bbl@setstring
2544      \let\bbl@stringdef\bbl@encstring
2545    \else        % ie, strings=value
2546      \bbl@sctest
2547      \ifin@
2548        \let\AfterBabelCommands\bbl@aftercmds
2549        \let\SetString\bbl@setstring
2550        \let\bbl@stringdef\bbl@provstring
2551    \fi\fi\fi
2552    \bbl@scswitch
2553    \ifx\bbl@G\@empty
2554      \def\SetString##1##2{%
2555        \bbl@error{Missing group for string \string##1}%
2556          {You must assign strings to some category, typically\\%
2557           captions or extras, but you set none}}%
2558    \fi
2559    \ifx\@empty#1%
2560      \bbl@usehooks{defaultcommands}{}%
2561    \else
2562      \@expandtwoargs
2563      \bbl@usehooks{encodedcommands}{{\bbl@sc@charset}{\bbl@sc@fontenc}}%
2564    \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) .

```
2565 \def\bbl@forlang#1#2{%
2566   \bbl@for#1\bbl@L{%
2567     \bbl@xin@{,#1,}{,\BabelLanguages,}%
2568     \ifin@#2\relax\fi}}
2569 \def\bbl@scswitch{%
2570   \bbl@forlang\bbl@tempa{%
2571     \ifx\bbl@G\@empty\else
2572       \ifx\SetString\@gobbletwo\else
2573         \edef\bbl@GL{\bbl@G\bbl@tempa}%
2574         \bbl@xin@{,\bbl@GL,}{,\bbl@screset,}%
2575         \ifin@\else
2576           \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
2577           \xdef\bbl@screset{\bbl@screset,\bbl@GL}%
2578         \fi
```

```
2579        \fi
2580      \fi}}
2581 \AtEndOfPackage{%
2582   \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{}{#2}}}%
2583   \let\bbl@scswitch\relax}
2584 \@onlypreamble\EndBabelCommands
2585 \def\EndBabelCommands{%
2586   \bbl@usehooks{stopcommands}{}%
2587   \endgroup
2588   \endgroup
2589   \bbl@scafter}
2590 \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.

```
2591 \def\bbl@setstring#1#2{% eg, \prefacename{<string>}
2592   \bbl@forlang\bbl@tempa{%
2593     \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
2594     \bbl@ifunset{\bbl@LC}% eg, \germanchaptername
2595       {\bbl@exp{%
2596          \global\\\bbl@add\<\bbl@G\bbl@tempa>{\\\bbl@scset\\#1\<\bbl@LC>}}}%
2597       {}%
2598     \def\BabelString{#2}%
2599     \bbl@usehooks{stringprocess}{}%
2600     \expandafter\bbl@stringdef
2601       \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`.

```
2602 \ifx\bbl@opt@strings\relax
2603   \def\bbl@scset#1#2{\def#1{\bbl@encoded#2}}
2604   \bbl@patchuclc
2605   \let\bbl@encoded\relax
2606   \def\bbl@encoded@uclc#1{%
2607     \@inmathwarn#1%
2608     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
2609       \expandafter\ifx\csname ?\string#1\endcsname\relax
2610         \TextSymbolUnavailable#1%
2611       \else
2612         \csname ?\string#1\endcsname
2613       \fi
2614     \else
2615       \csname\cf@encoding\string#1\endcsname
2616     \fi}
2617 \else
2618   \def\bbl@scset#1#2{\def#1{#2}}
2619 \fi
```

Define `\SetStringLoop`, which is actually set inside `\StartBabelCommands`. The current
definition is somewhat complicated because we need a count, but `\count@` is not under

our control (remember \SetString may call hooks). Instead of defining a dedicated count, we just "pre-expand" its value.

```
2620 ⟨⟨∗Macros local to BabelCommands⟩⟩ ≡
2621 \def\SetStringLoop##1##2{%
2622     \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
2623     \count@\z@
2624     \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
2625         \advance\count@\@ne
2626         \toks@\expandafter{\bbl@tempa}%
2627         \bbl@exp{%
2628             \\\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
2629             \count@=\the\count@\relax}}}%
2630 ⟨⟨/Macros local to BabelCommands⟩⟩
```

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

```
2631 \def\bbl@aftercmds#1{%
2632     \toks@\expandafter{\bbl@scafter#1}%
2633     \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.

```
2634 ⟨⟨∗Macros local to BabelCommands⟩⟩ ≡
2635     \newcommand\SetCase[3][]{%
2636         \bbl@patchuclc
2637         \bbl@forlang\bbl@tempa{%
2638             \expandafter\bbl@encstring
2639                 \csname\bbl@tempa @bbl@uclc\endcsname{\bbl@tempa##1}%
2640             \expandafter\bbl@encstring
2641                 \csname\bbl@tempa @bbl@uc\endcsname{##2}%
2642             \expandafter\bbl@encstring
2643                 \csname\bbl@tempa @bbl@lc\endcsname{##3}}}%
2644 ⟨⟨/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.

```
2645 ⟨⟨∗Macros local to BabelCommands⟩⟩ ≡
2646     \newcommand\SetHyphenMap[1]{%
2647         \bbl@forlang\bbl@tempa{%
2648             \expandafter\bbl@stringdef
2649                 \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
2650 ⟨⟨/Macros local to BabelCommands⟩⟩
```

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

```
2651 \newcommand\BabelLower[2]{% one to one.
2652     \ifnum\lccode#1=#2\else
2653         \babel@savevariable{\lccode#1}%
2654         \lccode#1=#2\relax
2655     \fi}
2656 \newcommand\BabelLowerMM[4]{% many-to-many
2657     \@tempcnta=#1\relax
2658     \@tempcntb=#4\relax
2659     \def\bbl@tempa{%
2660         \ifnum\@tempcnta>#2\else
2661             \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
```

```
2662        \advance\@tempcnta#3\relax
2663        \advance\@tempcntb#3\relax
2664        \expandafter\bbl@tempa
2665      \fi}%
2666    \bbl@tempa}
2667 \newcommand\BabelLowerMO[4]{% many-to-one
2668    \@tempcnta=#1\relax
2669    \def\bbl@tempa{%
2670      \ifnum\@tempcnta>#2\else
2671        \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
2672        \advance\@tempcnta#3
2673        \expandafter\bbl@tempa
2674      \fi}%
2675    \bbl@tempa}
```

The following package options control the behavior of hyphenation mapping.

```
2676 ⟨⟨∗More package options⟩⟩ ≡
2677 \DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\z@}
2678 \DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap\@ne}
2679 \DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\tw@}
2680 \DeclareOption{hyphenmap=other}{\chardef\bbl@opt@hyphenmap\thr@@}
2681 \DeclareOption{hyphenmap=other*}{\chardef\bbl@opt@hyphenmap4\relax}
2682 ⟨⟨/More package options⟩⟩
```

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

```
2683 \AtEndOfPackage{%
2684    \ifx\bbl@opt@hyphenmap\@undefined
2685      \bbl@xin@{,}{\bbl@language@opts}%
2686      \chardef\bbl@opt@hyphenmap\ifin@4\else\@ne\fi
2687    \fi}
```

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.

```
2688 \bbl@trace{Macros related to glyphs}
2689 \def\set@low@box#1{\setbox\tw@\hbox{,}\setbox\z@\hbox{#1}%
2690    \dimen\z@\ht\z@ \advance\dimen\z@ -\ht\tw@%
2691    \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.

```
2692 \def\save@sf@q#1{\leavevmode
2693    \begingroup
2694      \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
2695    \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.

```
2696 \ProvideTextCommand{\quotedblbase}{OT1}{%
2697    \save@sf@q{\set@low@box{\textquotedblright\/}%
2698       \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.

```
2699 \ProvideTextCommandDefault{\quotedblbase}{%
2700    \UseTextSymbol{OT1}{\quotedblbase}}
```

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

```
2701 \ProvideTextCommand{\quotesinglbase}{OT1}{%
2702    \save@sf@q{\set@low@box{\textquoteright\/}%
2703       \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.

```
2704 \ProvideTextCommandDefault{\quotesinglbase}{%
2705    \UseTextSymbol{OT1}{\quotesinglbase}}
```

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

```
2706 \ProvideTextCommand{\guillemetleft}{OT1}{%
2707    \ifmmode
2708       \ll
2709    \else
2710       \save@sf@q{\nobreak
2711          \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2712    \fi}
2713 \ProvideTextCommand{\guillemetright}{OT1}{%
2714    \ifmmode
2715       \gg
2716    \else
2717       \save@sf@q{\nobreak
2718          \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2719    \fi}
2720 \ProvideTextCommand{\guillemotleft}{OT1}{%
2721    \ifmmode
2722       \ll
2723    \else
2724       \save@sf@q{\nobreak
2725          \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2726    \fi}
2727 \ProvideTextCommand{\guillemotright}{OT1}{%
2728    \ifmmode
2729       \gg
2730    \else
2731       \save@sf@q{\nobreak
2732          \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2733    \fi}
```

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

```
2734 \ProvideTextCommandDefault{\guillemetleft}{%
2735    \UseTextSymbol{OT1}{\guillemetleft}}
2736 \ProvideTextCommandDefault{\guillemetright}{%
2737    \UseTextSymbol{OT1}{\guillemetright}}
2738 \ProvideTextCommandDefault{\guillemotleft}{%
2739    \UseTextSymbol{OT1}{\guillemotleft}}
```

```
2740 \ProvideTextCommandDefault{\guillemotright}{%
2741   \UseTextSymbol{OT1}{\guillemotright}}
```

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

\guilsinglright
```
2742 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2743   \ifmmode
2744     <%
2745   \else
2746     \save@sf@q{\nobreak
2747       \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
2748   \fi}
2749 \ProvideTextCommand{\guilsinglright}{OT1}{%
2750   \ifmmode
2751     >%
2752   \else
2753     \save@sf@q{\nobreak
2754       \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
2755   \fi}
```

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

```
2756 \ProvideTextCommandDefault{\guilsinglleft}{%
2757   \UseTextSymbol{OT1}{\guilsinglleft}}
2758 \ProvideTextCommandDefault{\guilsinglright}{%
2759   \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

\IJ   encoded fonts. Therefore we fake it for the OT1 encoding.

```
2760 \DeclareTextCommand{\ij}{OT1}{%
2761   i\kern-0.02em\bbl@allowhyphens j}
2762 \DeclareTextCommand{\IJ}{OT1}{%
2763   I\kern-0.02em\bbl@allowhyphens J}
2764 \DeclareTextCommand{\ij}{T1}{\char188}
2765 \DeclareTextCommand{\IJ}{T1}{\char156}
```

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

```
2766 \ProvideTextCommandDefault{\ij}{%
2767   \UseTextSymbol{OT1}{\ij}}
2768 \ProvideTextCommandDefault{\IJ}{%
2769   \UseTextSymbol{OT1}{\IJ}}
```

\dj   The croatian language needs the letters \dj and \DJ; they are available in the T1 encoding,

\DJ   but not in 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).

```
2770 \def\crrtic@{\hrule height0.1ex width0.3em}
2771 \def\crttic@{\hrule height0.1ex width0.33em}
2772 \def\ddj@{%
2773   \setbox0\hbox{d}\dimen@=\ht0
2774   \advance\dimen@1ex
2775   \dimen@.45\dimen@
2776   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2777   \advance\dimen@ii.5ex
2778   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
```

```
2779 \def\DDJ@{%
2780   \setbox0\hbox{D}\dimen@=.55\ht0
2781   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2782   \advance\dimen@ii.15ex %              correction for the dash position
2783   \advance\dimen@ii-.15\fontdimen7\font %      correction for cmtt font
2784   \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
2785   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
2786 %
2787 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2788 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}
```

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

```
2789 \ProvideTextCommandDefault{\dj}{%
2790   \UseTextSymbol{OT1}{\dj}}
2791 \ProvideTextCommandDefault{\DJ}{%
2792   \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.

```
2793 \DeclareTextCommand{\SS}{OT1}{SS}
2794 \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
```
2795 \ProvideTextCommandDefault{\glq}{%
2796   \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}
```

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

```
2797 \ProvideTextCommand{\grq}{T1}{%
2798   \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}
2799 \ProvideTextCommand{\grq}{TU}{%
2800   \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
2801 \ProvideTextCommand{\grq}{OT1}{%
2802   \save@sf@q{\kern-.0125em
2803     \textormath{\textquoteleft}{\mbox{\textquoteleft}}%
2804     \kern.07em\relax}}
2805 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}\grq}
```

\glqq   The 'german' double quotes.
\grqq
```
2806 \ProvideTextCommandDefault{\glqq}{%
2807   \textormath{\quotedblbase}{\mbox{\quotedblbase}}}
```

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

```
2808 \ProvideTextCommand{\grqq}{T1}{%
2809   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2810 \ProvideTextCommand{\grqq}{TU}{%
2811   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2812 \ProvideTextCommand{\grqq}{OT1}{%
```

```
2813     \save@sf@q{\kern-.07em
2814       \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}%
2815       \kern.07em\relax}}
2816 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}
```

The 'french' single guillemets.

```
2817 \ProvideTextCommandDefault{\flq}{%
2818     \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
2819 \ProvideTextCommandDefault{\frq}{%
2820     \textormath{\guilsinglright}{\mbox{\guilsinglright}}}
```

The 'french' double guillemets.

```
2821 \ProvideTextCommandDefault{\flqq}{%
2822     \textormath{\guillemetleft}{\mbox{\guillemetleft}}}
2823 \ProvideTextCommandDefault{\frqq}{%
2824     \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.

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

```
2825 \def\umlauthigh{%
2826     \def\bbl@umlauta##1{\leavevmode\bgroup%
2827         \expandafter\accent\csname\f@encoding dqpos\endcsname
2828         ##1\bbl@allowhyphens\egroup}%
2829     \let\bbl@umlaute\bbl@umlauta}
2830 \def\umlautlow{%
2831     \def\bbl@umlauta{\protect\lower@umlaut}}
2832 \def\umlautelow{%
2833     \def\bbl@umlaute{\protect\lower@umlaut}}
2834 \umlauthigh
```

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.

```
2835 \expandafter\ifx\csname U@D\endcsname\relax
2836     \csname newdimen\endcsname\U@D
2837 \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.

```
2838 \def\lower@umlaut#1{%
2839     \leavevmode\bgroup
2840         \U@D 1ex%
```

133

```
2841    {\setbox\z@\hbox{%
2842      \expandafter\char\csname\f@encoding dqpos\endcsname}%
2843      \dimen@ -.45ex\advance\dimen@\ht\z@
2844      \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2845    \expandafter\accent\csname\f@encoding dqpos\endcsname
2846    \fontdimen5\font\U@D #1%
2847    \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).

```
2848 \AtBeginDocument{%
2849    \DeclareTextCompositeCommand{\"}{OT1}{a}{\bbl@umlauta{a}}%
2850    \DeclareTextCompositeCommand{\"}{OT1}{e}{\bbl@umlaute{e}}%
2851    \DeclareTextCompositeCommand{\"}{OT1}{i}{\bbl@umlaute{\i}}%
2852    \DeclareTextCompositeCommand{\"}{OT1}{\i}{\bbl@umlaute{\i}}%
2853    \DeclareTextCompositeCommand{\"}{OT1}{o}{\bbl@umlauta{o}}%
2854    \DeclareTextCompositeCommand{\"}{OT1}{u}{\bbl@umlauta{u}}%
2855    \DeclareTextCompositeCommand{\"}{OT1}{A}{\bbl@umlauta{A}}%
2856    \DeclareTextCompositeCommand{\"}{OT1}{E}{\bbl@umlaute{E}}%
2857    \DeclareTextCompositeCommand{\"}{OT1}{I}{\bbl@umlaute{I}}%
2858    \DeclareTextCompositeCommand{\"}{OT1}{O}{\bbl@umlauta{O}}%
2859    \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.

```
2860 \ifx\l@english\@undefined
2861    \chardef\l@english\z@
2862 \fi
2863 % The following is used to cancel rules in ini files (see Amharic).
2864 \ifx\l@babelnohyhens\@undefined
2865    \newlanguage\l@babelnohyphens
2866 \fi
```

9.13 Layout

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

```
2867 \bbl@trace{Bidi layout}
2868 \providecommand\IfBabelLayout[3]{#3}%
2869 \newcommand\BabelPatchSection[1]{%
2870    \@ifundefined{#1}{}{%
2871      \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
2872      \@namedef{#1}{%
2873        \@ifstar{\bbl@presec@s{#1}}%
2874                {\@dblarg{\bbl@presec@x{#1}}}}}}
2875 \def\bbl@presec@x#1[#2]#3{%
2876    \bbl@exp{%
2877      \\\select@language@x{\bbl@main@language}%
2878      \\\bbl@cs{sspre@#1}%
2879      \\\bbl@cs{ss@#1}%
2880        [\\\foreignlanguage{\languagename}{\unexpanded{#2}}]%
2881        {\\\foreignlanguage{\languagename}{\unexpanded{#3}}}%
```

```
2882      \\\select@language@x{\languagename}}}
2883 \def\bbl@presec@s#1#2{%
2884   \bbl@exp{%
2885     \\\select@language@x{\bbl@main@language}%
2886     \\\bbl@cs{sspre@#1}%
2887     \\\bbl@cs{ss@#1}*%
2888       {\\\foreignlanguage{\languagename}{\unexpanded{#2}}}%
2889     \\\select@language@x{\languagename}}}
2890 \IfBabelLayout{sectioning}%
2891   {\BabelPatchSection{part}%
2892    \BabelPatchSection{chapter}%
2893    \BabelPatchSection{section}%
2894    \BabelPatchSection{subsection}%
2895    \BabelPatchSection{subsubsection}%
2896    \BabelPatchSection{paragraph}%
2897    \BabelPatchSection{subparagraph}%
2898    \def\babel@toc#1{%
2899      \select@language@x{\bbl@main@language}}}{}
2900 \IfBabelLayout{captions}%
2901   {\BabelPatchSection{caption}}{}
```

9.14   Load engine specific macros

```
2902 \bbl@trace{Input engine specific macros}
2903 \ifcase\bbl@engine
2904   \input txtbabel.def
2905 \or
2906   \input luababel.def
2907 \or
2908   \input xebabel.def
2909 \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.

```
2910 \bbl@trace{Creating languages and reading ini files}
2911 \newcommand\babelprovide[2][]{%
2912   \let\bbl@savelangname\languagename
2913   \edef\bbl@savelocaleid{\the\localeid}%
2914   % Set name and locale id
2915   \edef\languagename{#2}%
2916   % \global\@namedef{bbl@lcname@#2}{#2}%
2917   \bbl@id@assign
2918   \let\bbl@KVP@captions\@nil
2919   \let\bbl@KVP@date\@nil
2920   \let\bbl@KVP@import\@nil
2921   \let\bbl@KVP@main\@nil
2922   \let\bbl@KVP@script\@nil
2923   \let\bbl@KVP@language\@nil
2924   \let\bbl@KVP@hyphenrules\@nil
2925   \let\bbl@KVP@mapfont\@nil
2926   \let\bbl@KVP@maparabic\@nil
2927   \let\bbl@KVP@mapdigits\@nil
2928   \let\bbl@KVP@intraspace\@nil
2929   \let\bbl@KVP@intrapenalty\@nil
2930   \let\bbl@KVP@onchar\@nil
```

```
2931    \let\bbl@KVP@alph\@nil
2932    \let\bbl@KVP@Alph\@nil
2933    \let\bbl@KVP@labels\@nil
2934    \bbl@csarg\let{KVP@labels*}\@nil
2935    \bbl@forkv{#1}{%  TODO - error handling
2936      \in@{/}{##1}%
2937      \ifin@
2938        \bbl@renewinikey##1\@@{##2}%
2939      \else
2940        \bbl@csarg\def{KVP@##1}{##2}%
2941      \fi}%
2942    \let\bbl@saverenew@captions\bbl@renew@captions
2943    % == import, captions ==
2944    \ifx\bbl@KVP@import\@nil\else
2945      \bbl@exp{\\\bbl@ifblank{\bbl@KVP@import}}%
2946        {\ifx\bbl@initoload\relax
2947           \begingroup
2948             \def\BabelBeforeIni##1##2{\gdef\bbl@KVP@import{##1}\endinput}%
2949             \bbl@input@texini{#2}%
2950           \endgroup
2951         \else
2952           \xdef\bbl@KVP@import{\bbl@initoload}%
2953         \fi}%
2954        {}%
2955    \fi
2956    \ifx\bbl@KVP@captions\@nil
2957      \let\bbl@KVP@captions\bbl@KVP@import
2958    \fi
2959    % Load ini
2960    \bbl@ifunset{date#2}%
2961      {\bbl@provide@new{#2}}%
2962      {\bbl@ifblank{#1}%
2963        {\bbl@error
2964          {If you want to modify `#2' you must tell how in\\%
2965           the optional argument. See the manual for the\\%
2966           available options.}%
2967          {Use this macro as documented}}%
2968        {\bbl@provide@renew{#2}}}%
2969    % Post tasks
2970    \bbl@ifunset{bbl@extracaps@#2}%
2971      {\bbl@exp{\\\babelensure[exclude=\\\today]{#2}}}%
2972      {\toks@\expandafter\expandafter\expandafter
2973        {\csname bbl@extracaps@#2\endcsname}%
2974        \bbl@exp{\\\babelensure[exclude=\\\today,include=\the\toks@]{#2}}}%
2975    \bbl@ifunset{bbl@ensure@\languagename}%
2976      {\bbl@exp{%
2977        \\\DeclareRobustCommand\<bbl@ensure@\languagename>[1]{%
2978          \\\foreignlanguage{\languagename}%
2979          {####1}}}}%
2980      {}%
2981    \bbl@exp{%
2982      \\\bbl@toglobal\<bbl@ensure@\languagename>%
2983      \\\bbl@toglobal\<bbl@ensure@\languagename\space>}%
2984    % At this point all parameters are defined if 'import'. Now we
2985    % execute some code depending on them. But what about if nothing was
2986    % imported? We just load the very basic parameters.
2987    \bbl@load@basic{#2}%
2988    % == script, language ==
2989    % Override the values from ini or defines them
```

```
2990    \ifx\bbl@KVP@script\@nil\else
2991      \bbl@csarg\edef{sname@#2}{\bbl@KVP@script}%
2992    \fi
2993    \ifx\bbl@KVP@language\@nil\else
2994      \bbl@csarg\edef{lname@#2}{\bbl@KVP@language}%
2995    \fi
2996    % == onchar ==
2997    \ifx\bbl@KVP@onchar\@nil\else
2998      \bbl@luahyphenate
2999      \directlua{
3000        if Babel.locale_mapped == nil then
3001          Babel.locale_mapped = true
3002          Babel.linebreaking.add_before(Babel.locale_map)
3003          Babel.loc_to_scr = {}
3004          Babel.chr_to_loc = Babel.chr_to_loc or {}
3005        end}%
3006      \bbl@xin@{ ids }{ \bbl@KVP@onchar\space}%
3007      \ifin@
3008        \ifx\bbl@starthyphens\@undefined % Needed if no explicit selection
3009          \AddBabelHook{babel-onchar}{beforestart}{{\bbl@starthyphens}}%
3010        \fi
3011        \bbl@exp{\\\bbl@add\\\bbl@starthyphens
3012          {\\\bbl@patterns@lua{\languagename}}}%
3013        % TODO - error/warning if no script
3014        \directlua{
3015          if Babel.script_blocks['\bbl@cl{sbcp}'] then
3016            Babel.loc_to_scr[\the\localeid] =
3017              Babel.script_blocks['\bbl@cl{sbcp}']
3018            Babel.locale_props[\the\localeid].lc = \the\localeid\space
3019            Babel.locale_props[\the\localeid].lg = \the\@nameuse{l@\languagename}\space
3020          end
3021        }%
3022      \fi
3023      \bbl@xin@{ fonts }{ \bbl@KVP@onchar\space}%
3024      \ifin@
3025        \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
3026        \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
3027        \directlua{
3028          if Babel.script_blocks['\bbl@cl{sbcp}'] then
3029            Babel.loc_to_scr[\the\localeid] =
3030              Babel.script_blocks['\bbl@cl{sbcp}']
3031          end}%
3032        \ifx\bbl@mapselect\@undefined
3033          \AtBeginDocument{%
3034            \expandafter\bbl@add\csname selectfont \endcsname{{\bbl@mapselect}}%
3035            {\selectfont}}%
3036          \def\bbl@mapselect{%
3037            \let\bbl@mapselect\relax
3038            \edef\bbl@prefontid{\fontid\font}}%
3039          \def\bbl@mapdir##1{%
3040            {\def\languagename{##1}%
3041             \let\bbl@ifrestoring\@firstoftwo % To avoid font warning
3042             \bbl@switchfont
3043             \directlua{
3044               Babel.locale_props[\the\csname bbl@id@@##1\endcsname]%
3045                        ['/\bbl@prefontid'] = \fontid\font\space}}}%
3046        \fi
3047        \bbl@exp{\\\bbl@add\\\bbl@mapselect{\\\bbl@mapdir{\languagename}}}%
3048      \fi
```

137

```
3049     % TODO - catch non-valid values
3050   \fi
3051 % == mapfont ==
3052 % For bidi texts, to switch the font based on direction
3053 \ifx\bbl@KVP@mapfont\@nil\else
3054   \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}{}%
3055     {\bbl@error{Option `\bbl@KVP@mapfont' unknown for\\%
3056                 mapfont. Use `direction'.%
3057                 {See the manual for details.}}}%
3058   \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
3059   \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
3060   \ifx\bbl@mapselect\@undefined
3061     \AtBeginDocument{%
3062       \expandafter\bbl@add\csname selectfont \endcsname{{\bbl@mapselect}}%
3063       {\selectfont}}%
3064     \def\bbl@mapselect{%
3065       \let\bbl@mapselect\relax
3066       \edef\bbl@prefontid{\fontid\font}}%
3067     \def\bbl@mapdir##1{%
3068       {\def\languagename{##1}%
3069        \let\bbl@ifrestoring\@firstoftwo % avoid font warning
3070        \bbl@switchfont
3071        \directlua{Babel.fontmap
3072          [\the\csname bbl@wdir@##1\endcsname]%
3073          [\bbl@prefontid]=\fontid\font}}}%
3074   \fi
3075   \bbl@exp{\\\bbl@add\\\bbl@mapselect{\\\bbl@mapdir{\languagename}}}%
3076 \fi
3077 % == Line breaking: intraspace, intrapenalty ==
3078 % For CJK, East Asian, Southeast Asian, if interspace in ini
3079 \ifx\bbl@KVP@intraspace\@nil\else % We can override the ini or set
3080   \bbl@csarg\edef{intsp@#2}{\bbl@KVP@intraspace}%
3081 \fi
3082 \bbl@provide@intraspace
3083 % == Line breaking: hyphenate.other.locale ==
3084 \bbl@ifunset{bbl@hyotl@\languagename}{}%
3085   {\bbl@csarg\bbl@replace{hyotl@\languagename}{ }{,}%
3086    \bbl@startcommands*{\languagename}{}%
3087      \bbl@csarg\bbl@foreach{hyotl@\languagename}{%
3088        \ifcase\bbl@engine
3089          \ifnum##1<257
3090            \SetHyphenMap{\BabelLower{##1}{##1}}%
3091          \fi
3092        \else
3093          \SetHyphenMap{\BabelLower{##1}{##1}}%
3094        \fi}%
3095    \bbl@endcommands}%
3096 % == Line breaking: hyphenate.other.script ==
3097 \bbl@ifunset{bbl@hyots@\languagename}{}%
3098   {\bbl@csarg\bbl@replace{hyots@\languagename}{ }{,}%
3099    \bbl@csarg\bbl@foreach{hyots@\languagename}{%
3100      \ifcase\bbl@engine
3101        \ifnum##1<257
3102          \global\lccode##1=##1\relax
3103        \fi
3104      \else
3105        \global\lccode##1=##1\relax
3106      \fi}}%
3107 % == Counters: maparabic ==
```

138

```
3108  % Native digits, if provided in ini (TeX level, xe and lua)
3109  \ifcase\bbl@engine\else
3110    \bbl@ifunset{bbl@dgnat@\languagename}{}%
3111      {\expandafter\ifx\csname bbl@dgnat@\languagename\endcsname\@empty\else
3112        \expandafter\expandafter\expandafter
3113        \bbl@setdigits\csname bbl@dgnat@\languagename\endcsname
3114        \ifx\bbl@KVP@maparabic\@nil\else
3115          \ifx\bbl@latinarabic\@undefined
3116            \expandafter\let\expandafter\@arabic
3117              \csname bbl@counter@\languagename\endcsname
3118          \else    % ie, if layout=counters, which redefines \@arabic
3119            \expandafter\let\expandafter\bbl@latinarabic
3120              \csname bbl@counter@\languagename\endcsname
3121          \fi
3122        \fi
3123      \fi}%
3124  \fi
3125  % == Counters: mapdigits ==
3126  % Native digits (lua level).
3127  \ifodd\bbl@engine
3128    \ifx\bbl@KVP@mapdigits\@nil\else
3129      \bbl@ifunset{bbl@dgnat@\languagename}{}%
3130        {\RequirePackage{luatexbase}%
3131         \bbl@activate@preotf
3132         \directlua{
3133           Babel = Babel or {}  %%% -> presets in luababel
3134           Babel.digits_mapped = true
3135           Babel.digits = Babel.digits or {}
3136           Babel.digits[\the\localeid] =
3137             table.pack(string.utfvalue('\bbl@cl{dgnat}'))
3138           if not Babel.numbers then
3139             function Babel.numbers(head)
3140               local LOCALE = luatexbase.registernumber'bbl@attr@locale'
3141               local GLYPH = node.id'glyph'
3142               local inmath = false
3143               for item in node.traverse(head) do
3144                 if not inmath and item.id == GLYPH then
3145                   local temp = node.get_attribute(item, LOCALE)
3146                   if Babel.digits[temp] then
3147                     local chr = item.char
3148                     if chr > 47 and chr < 58 then
3149                       item.char = Babel.digits[temp][chr-47]
3150                     end
3151                   end
3152                 elseif item.id == node.id'math' then
3153                   inmath = (item.subtype == 0)
3154                 end
3155               end
3156               return head
3157             end
3158           end
3159        }}%
3160    \fi
3161  \fi
3162  % == Counters: alph, Alph ==
3163  % What if extras<lang> contains a \babel@save\@alph? It won't be
3164  % restored correctly when exiting the language, so we ignore
3165  % this change with the \bbl@alph@saved trick.
3166  \ifx\bbl@KVP@alph\@nil\else
```

```
3167      \toks@\expandafter\expandafter\expandafter{%
3168        \csname extras\languagename\endcsname}%
3169      \bbl@exp{%
3170        \def\<extras\languagename>{%
3171          \let\\\bbl@alph@saved\\\@alph
3172          \the\toks@
3173          \let\\\@alph\\\bbl@alph@saved
3174          \\\babel@save\\\@alph
3175          \let\\\@alph\<bbl@cntr@\bbl@KVP@alph @\languagename>}}%
3176    \fi
3177    \ifx\bbl@KVP@Alph\@nil\else
3178      \toks@\expandafter\expandafter\expandafter{%
3179        \csname extras\languagename\endcsname}%
3180      \bbl@exp{%
3181        \def\<extras\languagename>{%
3182          \let\\\bbl@Alph@saved\\\@Alph
3183          \the\toks@
3184          \let\\\@Alph\\\bbl@Alph@saved
3185          \\\babel@save\\\@Alph
3186          \let\\\@Alph\<bbl@cntr@\bbl@KVP@Alph @\languagename>}}%
3187    \fi
3188    % == require.babel in ini ==
3189    % To load or reaload the babel-*.tex, if require.babel in ini
3190    \ifx\bbl@beforestart\relax\else  % But only in preamble
3191      \bbl@ifunset{bbl@rqtex@\languagename}{}%
3192        {\expandafter\ifx\csname bbl@rqtex@\languagename\endcsname\@empty\else
3193          \let\BabelBeforeIni\@gobbletwo
3194          \chardef\atcatcode=\catcode`\@
3195          \catcode`\@=11\relax
3196          \bbl@input@texini{\bbl@cs{rqtex@\languagename}}%
3197          \catcode`\@=\atcatcode
3198          \let\atcatcode\relax
3199        \fi}%
3200    \fi
3201    % == caption redefinition ==
3202    \ifx\bbl@KVP@captions\@nil
3203      \def\bbl@elt##1##2{%
3204        \bbl@ifunset{\languagename ##1name}%
3205          {\toks@{##2}%
3206           \bbl@exp{%
3207             \\\bbl@add\<captions\languagename>{\def\<##1name>{\the\toks@}}}}%
3208          {\@namedef{\languagename##1name}{##2}}}%
3209      \@nameuse{bbl@saverenew@captions}%
3210    \fi
3211    % == main ==
3212    \ifx\bbl@KVP@main\@nil  % Restore only if not 'main'
3213      \let\languagename\bbl@savelangname
3214      \chardef\localeid\bbl@savelocaleid\relax
3215    \fi}
```

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

```
3216 \def\bbl@provide@new#1{%
3217  \@namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
3218  \@namedef{extras#1}{}%
3219  \@namedef{noextras#1}{}%
3220  \bbl@startcommands*{#1}{captions}%
3221    \ifx\bbl@KVP@captions\@nil %      and also if import, implicit
3222      \def\bbl@tempb##1{%              elt for \bbl@captionslist
3223        \ifx##1\@empty\else
```

140

```
3224        \bbl@exp{%
3225          \\\SetString\\##1{%
3226            \\\bbl@nocaption{\bbl@stripslash##1}{#1\bbl@stripslash##1}}}%
3227          \expandafter\bbl@tempb
3228        \fi}%
3229      \expandafter\bbl@tempb\bbl@captionslist\@empty
3230    \else
3231      \ifx\bbl@initoload\relax
3232        \bbl@read@ini{\bbl@KVP@captions}0%  Here letters cat = 11
3233      \else
3234        \bbl@read@ini{\bbl@initoload}0%  Here all letters cat = 11
3235      \fi
3236      \bbl@after@ini
3237      \bbl@savestrings
3238    \fi
3239  \StartBabelCommands*{#1}{date}%
3240    \ifx\bbl@KVP@import\@nil
3241      \bbl@exp{%
3242        \\\SetString\\\today{\\\bbl@nocaption{today}{#1today}}}%
3243    \else
3244      \bbl@savetoday
3245      \bbl@savedate
3246    \fi
3247  \bbl@endcommands
3248  \bbl@load@basic{#1}%
3249  % == hyphenmins == (only if new)
3250  \bbl@exp{%
3251    \gdef\<#1hyphenmins>{%
3252      {\bbl@ifunset{bbl@lfthm@#1}{2}{\bbl@cs{lfthm@#1}}}%
3253      {\bbl@ifunset{bbl@rgthm@#1}{3}{\bbl@cs{rgthm@#1}}}}}%
3254  % == hyphenrules ==
3255  \bbl@provide@hyphens{#1}%
3256  % == frenchspacing == (only if new)
3257  \bbl@ifunset{bbl@frspc@#1}{}%
3258    {\edef\bbl@tempa{\bbl@cl{frspc}}%
3259     \edef\bbl@tempa{\expandafter\@car\bbl@tempa\@nil}%
3260     \if u\bbl@tempa           % do nothing
3261     \else\if n\bbl@tempa      % non french
3262       \expandafter\bbl@add\csname extras#1\endcsname{%
3263         \let\bbl@elt\bbl@fs@elt@i
3264         \bbl@fs@chars}%
3265     \else\if y\bbl@tempa       % french
3266       \expandafter\bbl@add\csname extras#1\endcsname{%
3267         \let\bbl@elt\bbl@fs@elt@ii
3268         \bbl@fs@chars}%
3269     \fi\fi\fi}%
3270  %
3271  \ifx\bbl@KVP@main\@nil\else
3272    \expandafter\main@language\expandafter{#1}%
3273  \fi}
3274 % A couple of macros used above, to avoid hashes #######...
3275 \def\bbl@fs@elt@i#1#2#3{%
3276   \ifnum\sfcode`#1=#2\relax
3277     \babel@savevariable{\sfcode`#1}%
3278     \sfcode`#1=#3\relax
3279   \fi}%
3280 \def\bbl@fs@elt@ii#1#2#3{%
3281   \ifnum\sfcode`#1=#3\relax
3282     \babel@savevariable{\sfcode`#1}%
```

```
3283      \sfcode`#1=#2\relax
3284    \fi}%
3285 %
3286 \def\bbl@provide@renew#1{%
3287    \ifx\bbl@KVP@captions\@nil\else
3288      \StartBabelCommands*{#1}{captions}%
3289        \bbl@read@ini{\bbl@KVP@captions}0%    Here all letters cat = 11
3290        \bbl@after@ini
3291        \bbl@savestrings
3292      \EndBabelCommands
3293    \fi
3294    \ifx\bbl@KVP@import\@nil\else
3295      \StartBabelCommands*{#1}{date}%
3296        \bbl@savetoday
3297        \bbl@savedate
3298      \EndBabelCommands
3299    \fi
3300    % == hyphenrules ==
3301    \bbl@provide@hyphens{#1}}
3302 % Load the basic parameters (ids, typography, counters, and a few
3303 % more), while captions and dates are left out. But it may happen some
3304 % data has been loaded before automatically, so we first discard the
3305 % saved values.
3306 \def\bbl@linebreak@export{%
3307    \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
3308    \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
3309    \bbl@exportkey{lfthm}{typography.lefthyphenmin}{2}%
3310    \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
3311    \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
3312    \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
3313    \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
3314    \bbl@exportkey{intsp}{typography.intraspace}{}%
3315    \bbl@exportkey{chrng}{characters.ranges}{}}
3316 \def\bbl@load@basic#1{%
3317    \bbl@ifunset{bbl@inidata@\languagename}{}%
3318      {\getlocaleproperty\bbl@tempa{\languagename}{identification/load.level}%
3319        \ifcase\bbl@tempa\else
3320          \bbl@csarg\let{lname@\languagename}\relax
3321        \fi}%
3322    \bbl@ifunset{bbl@lname@#1}%
3323      {\def\BabelBeforeIni##1##2{%
3324        \begingroup
3325          \let\bbl@ini@captions@aux\@gobbletwo
3326          \def\bbl@inidate ####1.####2.####3.####4\relax ####5####6{}%
3327          \bbl@read@ini{##1}0%
3328          \bbl@linebreak@export
3329          \bbl@exportkey{dgnat}{numbers.digits.native}{}%
3330          \bbl@exportkey{frspc}{typography.frenchspacing}{u}% unset
3331          \ifx\bbl@initoload\relax\endinput\fi
3332        \endgroup}%
3333      \begingroup        % boxed, to avoid extra spaces:
3334        \ifx\bbl@initoload\relax
3335          \bbl@input@texini{#1}%
3336        \else
3337          \setbox\z@\hbox{\BabelBeforeIni{\bbl@initoload}{}}%
3338        \fi
3339      \endgroup}%
3340      {}}
```

The hyphenrules option is handled with an auxiliary macro.

```
3341 \def\bbl@provide@hyphens#1{%
3342   \let\bbl@tempa\relax
3343   \ifx\bbl@KVP@hyphenrules\@nil\else
3344     \bbl@replace\bbl@KVP@hyphenrules{ }{,}%
3345     \bbl@foreach\bbl@KVP@hyphenrules{%
3346       \ifx\bbl@tempa\relax    % if not yet found
3347         \bbl@ifsamestring{##1}{+}%
3348           {{\bbl@exp{\\\addlanguage\<l@##1>}}}%
3349           {}%
3350         \bbl@ifunset{l@##1}%
3351           {}%
3352           {\bbl@exp{\let\bbl@tempa\<l@##1>}}%
3353       \fi}%
3354   \fi
3355   \ifx\bbl@tempa\relax %        if no opt or no language in opt found
3356     \ifx\bbl@KVP@import\@nil
3357       \ifx\bbl@initoload\relax\else
3358         \bbl@exp{%                  and hyphenrules is not empty
3359           \\\bbl@ifblank{\bbl@cs{hyphr@#1}}%
3360             {}%
3361             {\let\\\bbl@tempa\<l@\bbl@cl{hyphr}>}}%
3362       \fi
3363     \else % if importing
3364       \bbl@exp{%                  and hyphenrules is not empty
3365         \\\bbl@ifblank{\bbl@cs{hyphr@#1}}%
3366           {}%
3367           {\let\\\bbl@tempa\<l@\bbl@cl{hyphr}>}}%
3368     \fi
3369   \fi
3370   \bbl@ifunset{bbl@tempa}%       ie, relax or undefined
3371     {\bbl@ifunset{l@#1}%          no hyphenrules found - fallback
3372       {\bbl@exp{\\\adddialect\<l@#1>\language}}%
3373       {}}%                        so, l@<lang> is ok - nothing to do
3374     {\bbl@exp{\\\adddialect\<l@#1>\bbl@tempa}}}% found in opt list or ini
3375
```

The reader of ini files. There are 3 possible cases: a section name (in the form [...]), a comment (starting with ;) and a key/value pair.

```
3376 \ifx\bbl@readstream\@undefined
3377   \csname newread\endcsname\bbl@readstream
3378 \fi
3379 \def\bbl@input@texini#1{%
3380   \bbl@bsphack
3381     \bbl@exp{%
3382       \catcode`\\\%=14 \catcode`\\\\=0
3383       \catcode`\\\{=1  \catcode`\\\}=2
3384       \lowercase{\\\InputIfFileExists{babel-#1.tex}{}{}}%
3385       \catcode`\\\%=\the\catcode`\%\relax
3386       \catcode`\\\\=\the\catcode`\\\relax
3387       \catcode`\\\{=\the\catcode`\{\relax
3388       \catcode`\\\}=\the\catcode`\}\relax}%
3389   \bbl@esphack}
3390 \def\bbl@inipreread#1=#2\@@{%
3391   \bbl@trim@def\bbl@tempa{#1}% Redundant below !!
3392   \bbl@trim\toks@{#2}%
3393   % Move trims here ??
3394   \bbl@ifunset{bbl@KVP@\bbl@section/\bbl@tempa}%
```

```
3395        {\bbl@exp{%
3396            \\\g@addto@macro\\\bbl@inidata{%
3397              \\\bbl@elt{\bbl@section}{\bbl@tempa}{\the\toks@}}}%
3398         \expandafter\bbl@inireader\bbl@tempa=#2\@@}%
3399        {}}%
3400 \def\bbl@fetch@ini#1#2{%
3401    \bbl@exp{\def\\\bbl@inidata{%
3402        \\\bbl@elt{identification}{tag.ini}{#1}%
3403        \\\bbl@elt{identification}{load.level}{#2}}}%
3404    \openin\bbl@readstream=babel-#1.ini
3405    \ifeof\bbl@readstream
3406      \bbl@error
3407        {There is no ini file for the requested language\\%
3408         (#1). Perhaps you misspelled it or your installation\\%
3409         is not complete.}%
3410        {Fix the name or reinstall babel.}%
3411    \else
3412      \catcode`\[=12 \catcode`\]=12 \catcode`\==12 \catcode`\&=12
3413      \catcode`\;=12 \catcode`\|=12 \catcode`\%=14
3414      \bbl@info{Importing
3415                  \ifcase#2 \or font and identification \or basic \fi
3416                  data for \languagename\\%
3417                from babel-#1.ini. Reported}%
3418      \loop
3419      \if T\ifeof\bbl@readstream F\fi T\relax % Trick, because inside \loop
3420        \endlinechar\m@ne
3421        \read\bbl@readstream to \bbl@line
3422        \endlinechar`\^^M
3423        \ifx\bbl@line\@empty\else
3424          \expandafter\bbl@iniline\bbl@line\bbl@iniline
3425        \fi
3426      \repeat
3427    \fi}
3428 \def\bbl@read@ini#1#2{%
3429    \bbl@csarg\edef{lini@\languagename}{#1}%
3430    \let\bbl@section\@empty
3431    \let\bbl@savestrings\@empty
3432    \let\bbl@savetoday\@empty
3433    \let\bbl@savedate\@empty
3434    \let\bbl@inireader\bbl@iniskip
3435    \bbl@fetch@ini{#1}{#2}%
3436    \bbl@foreach\bbl@renewlist{%
3437      \bbl@ifunset{bbl@renew@##1}{}{\bbl@inisec[##1]\@@}}%
3438    \global\let\bbl@renewlist\@empty
3439    % Ends last section. See \bbl@inisec
3440    \def\bbl@elt##1##2{\bbl@inireader##1=##2\@@}%
3441    \bbl@cs{renew@\bbl@section}%
3442    \global\bbl@csarg\let{renew@\bbl@section}\relax
3443    \bbl@cs{secpost@\bbl@section}%
3444    \bbl@csarg{\global\expandafter\let}{inidata@\languagename}\bbl@inidata
3445    \bbl@exp{\\\bbl@add@list\\\bbl@ini@loaded{\languagename}}%
3446    \bbl@toglobal\bbl@ini@loaded}
3447 \def\bbl@iniline#1\bbl@iniline{%
3448    \@ifnextchar[\bbl@inisec{\@ifnextchar;\bbl@iniskip\bbl@inipreread}#1\@@}% ]
```

The special cases for comment lines and sections are handled by the two following commands. In sections, we provide the posibility to take extra actions at the end or at the start. By default, key=val pairs are ignored. The secpost "hook" is used only by 'identification', while secpre only by date.gregorian.licr.

```
3449 \def\bbl@iniskip#1\@@{}%        if starts with ;
3450 \def\bbl@inisec[#1]#2\@@{%      if starts with opening bracket
3451   \def\bbl@elt##1##2{%
3452     \expandafter\toks@\expandafter{%
3453       \expandafter{\bbl@section}{##1}{##2}}%
3454     \bbl@exp{%
3455       \\\g@addto@macro\\\bbl@inidata{\\\bbl@elt\the\toks@}}%
3456     \bbl@inireader##1=##2\@@}%
3457   \bbl@cs{renew@\bbl@section}%
3458   \global\bbl@csarg\let{renew@\bbl@section}\relax
3459   \bbl@cs{secpost@\bbl@section}%
3460   % The previous code belongs to the previous section.
3461   % -------------------------
3462   % Now start the current one.
3463   \in@{=date.}{=#1}%
3464   \ifin@
3465     \lowercase{\def\bbl@tempa{=#1=}}%
3466     \bbl@replace\bbl@tempa{=date.gregorian}{}%
3467     \bbl@replace\bbl@tempa{=date.}{}%
3468     \in@{.licr=}{#1=}%
3469     \ifin@
3470       \ifcase\bbl@engine
3471         \bbl@replace\bbl@tempa{.licr=}{}%
3472       \else
3473         \let\bbl@tempa\relax
3474       \fi
3475     \fi
3476     \ifx\bbl@tempa\relax\else
3477       \bbl@replace\bbl@tempa{=}{}%
3478       \bbl@exp{%
3479         \def\<bbl@inikv@#1>####1=####2\\\@@{%
3480           \\\bbl@inidate####1...\relax{####2}{\bbl@tempa}}}%
3481     \fi
3482   \fi
3483   \def\bbl@section{#1}%
3484   \def\bbl@elt##1##2{%
3485     \@namedef{bbl@KVP@#1/##1}{}}%
3486   \bbl@cs{renew@#1}%
3487   \bbl@cs{secpre@#1}%  pre-section `hook'
3488   \bbl@ifunset{bbl@inikv@#1}%
3489     {\let\bbl@inireader\bbl@iniskip}%
3490     {\bbl@exp{\let\\\bbl@inireader\<bbl@inikv@#1>}}}
3491 \let\bbl@renewlist\@empty
3492 \def\bbl@renewinikey#1/#2\@@#3{%
3493   \bbl@ifunset{bbl@renew@#1}%
3494     {\bbl@add@list\bbl@renewlist{#1}}%
3495     {}%
3496   \bbl@csarg\bbl@add{renew@#1}{\bbl@elt{#2}{#3}}}
```

Reads a key=val line and stores the trimmed val in \bbl@@kv@<section>.<key>.

```
3497 \def\bbl@inikv#1=#2\@@{%        key=value
3498   \bbl@trim@def\bbl@tempa{#1}%
3499   \bbl@trim\toks@{#2}%
3500   \bbl@csarg\edef{@kv@\bbl@section.\bbl@tempa}{\the\toks@}}
```

The previous assignments are local, so we need to export them. If the value is empty, we can provide a default value.

```
3501 \def\bbl@exportkey#1#2#3{%
3502   \bbl@ifunset{bbl@@kv@#2}%
```

```
3503        {\bbl@csarg\gdef{#1@\languagename}{#3}}%
3504        {\expandafter\ifx\csname bbl@@kv@#2\endcsname\@empty
3505           \bbl@csarg\gdef{#1@\languagename}{#3}%
3506         \else
3507           \bbl@exp{\global\let\<bbl@#1@\languagename>\<bbl@@kv@#2>}%
3508         \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@secpost@identification is called always (via \bbl@inisec), while \bbl@after@ini must be called explicitly after \bbl@read@ini if necessary.

```
3509 \def\bbl@iniwarning#1{%
3510   \bbl@ifunset{bbl@@kv@identification.warning#1}{}%
3511     {\bbl@warning{%
3512        From babel-\bbl@cs{lini@\languagename}.ini:\\%
3513        \bbl@cs{@kv@identification.warning#1}\\%
3514        Reported }}}
3515 %
3516 \let\bbl@inikv@identification\bbl@inikv
3517 \def\bbl@secpost@identification{%
3518   \bbl@iniwarning{}%
3519   \ifcase\bbl@engine
3520     \bbl@iniwarning{.pdflatex}%
3521   \or
3522     \bbl@iniwarning{.lualatex}%
3523   \or
3524     \bbl@iniwarning{.xelatex}%
3525   \fi%
3526   \bbl@exportkey{elname}{identification.name.english}{}%
3527   \bbl@exp{\\\bbl@exportkey{lname}{identification.name.opentype}%
3528     {\csname bbl@elname@\languagename\endcsname}}%
3529   \bbl@exportkey{lbcp}{identification.tag.bcp47}{}% TODO
3530   \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
3531   \bbl@exportkey{esname}{identification.script.name}{}%
3532   \bbl@exp{\\\bbl@exportkey{sname}{identification.script.name.opentype}%
3533     {\csname bbl@esname@\languagename\endcsname}}%
3534   \bbl@exportkey{sbcp}{identification.script.tag.bcp47}{}%
3535   \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
3536   \ifbbl@bcptoname
3537     \bbl@csarg\xdef{bcp@map@\bbl@cl{lbcp}}{\languagename}%
3538   \fi}
```

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

```
3539 \let\bbl@inikv@typography\bbl@inikv
3540 \let\bbl@inikv@characters\bbl@inikv
3541 \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'.

```
3542 \def\bbl@inikv@counters#1=#2\@@{%
3543   \bbl@ifsamestring{#1}{digits}%
3544     {\bbl@error{The counter name 'digits' is reserved for mapping\\%
3545                decimal digits}%
3546               {Use another name.}}%
3547     {}%
3548   \def\bbl@tempc{#1}%
3549   \bbl@trim@def{\bbl@tempb*}{#2}%
3550   \in@{.1$}{#1$}%
```

146

```
3551  \ifin@
3552    \bbl@replace\bbl@tempc{.1}{}%
3553    \bbl@csarg\protected@xdef{cntr@\bbl@tempc @\languagename}{%
3554      \noexpand\bbl@alphnumeral{\bbl@tempc}}%
3555  \fi
3556  \in@{.F.}{#1}%
3557  \ifin@\else\in@{.S.}{#1}\fi
3558  \ifin@
3559    \bbl@csarg\protected@xdef{cntr@#1@\languagename}{\bbl@tempb*}%
3560  \else
3561    \toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa
3562    \expandafter\bbl@buildifcase\bbl@tempb* \\ % Space after \\
3563    \bbl@csarg{\global\expandafter\let}{cntr@#1@\languagename}\bbl@tempa
3564  \fi}
3565 \def\bbl@after@ini{%
3566   \bbl@linebreak@export
3567   \bbl@exportkey{dgnat}{numbers.digits.native}{}%
3568   \bbl@exportkey{rqtex}{identification.require.babel}{}%
3569   \bbl@exportkey{frspc}{typography.frenchspacing}{u}% unset
3570   \bbl@toglobal\bbl@savetoday
3571   \bbl@toglobal\bbl@savedate}
```

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.

```
3572 \ifcase\bbl@engine
3573   \bbl@csarg\def{inikv@captions.licr}#1=#2\@@{%
3574     \bbl@ini@captions@aux{#1}{#2}}
3575 \else
3576   \def\bbl@inikv@captions#1=#2\@@{%
3577     \bbl@ini@captions@aux{#1}{#2}}
3578 \fi
```

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

```
3579 \def\bbl@ini@captions@aux#1#2{%
3580   \bbl@trim@def\bbl@tempa{#1}%
3581   \bbl@xin@{.template}{\bbl@tempa}%
3582   \ifin@
3583     \bbl@replace\bbl@tempa{.template}{}%
3584     \def\bbl@toreplace{#2}%
3585     \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
3586     \bbl@replace\bbl@toreplace{[[}{\csname}%
3587     \bbl@replace\bbl@toreplace{[}{\csname the}%
3588     \bbl@replace\bbl@toreplace{]]}{name\endcsname{}}%
3589     \bbl@replace\bbl@toreplace{]}{\endcsname{}}%
3590     \bbl@xin@{,\bbl@tempa,}{,chapter,}%
3591     \ifin@
3592       \bbl@patchchapter
3593       \global\bbl@csarg\let{chapfmt@\languagename}\bbl@toreplace
3594     \fi
3595     \bbl@xin@{,\bbl@tempa,}{,appendix,}%
3596     \ifin@
3597       \bbl@patchchapter
3598       \global\bbl@csarg\let{appxfmt@\languagename}\bbl@toreplace
3599     \fi
3600     \bbl@xin@{,\bbl@tempa,}{,part,}%
3601     \ifin@
3602       \bbl@patchpart
3603       \global\bbl@csarg\let{partfmt@\languagename}\bbl@toreplace
```

```
3604      \fi
3605      \bbl@xin@{,\bbl@tempa,}{,figure,table,}%
3606      \ifin@
3607        \toks@\expandafter{\bbl@toreplace}%
3608        \bbl@exp{\gdef\<fnum@\bbl@tempa>{\the\toks@}}%
3609      \fi
3610    \else
3611      \bbl@ifblank{#2}%
3612        {\bbl@exp{%
3613           \toks@{\\\bbl@nocaption{\bbl@tempa}{\languagename\bbl@tempa name}}}}%
3614        {\bbl@trim\toks@{#2}}%
3615      \bbl@exp{%
3616        \\\bbl@add\\\bbl@savestrings{%
3617          \\\SetString\<\bbl@tempa name>{\the\toks@}}}%
3618      \toks@\expandafter{\bbl@captionslist}%
3619      \bbl@exp{\\\in@{\<\bbl@tempa name>}{\the\toks@}}%
3620      \ifin@\else
3621        \bbl@exp{%
3622          \\\bbl@add\<bbl@extracaps@\languagename>{\<\bbl@tempa name>}%
3623          \\\bbl@toglobal\<bbl@extracaps@\languagename>}%
3624      \fi
3625    \fi}
```

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

```
3626 \def\bbl@list@the{%
3627    part,chapter,section,subsection,subsubsection,paragraph,%
3628    subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,%
3629    table,page,footnote,mpfootnote,mpfn}
3630 \def\bbl@map@cnt#1{%  #1:roman,etc, // #2:enumi,etc
3631    \bbl@ifunset{bbl@map@#1@\languagename}%
3632      {\@nameuse{#1}}%
3633      {\@nameuse{bbl@map@#1@\languagename}}}
3634 \def\bbl@inikv@labels#1=#2\@@{%
3635    \in@{.map}{#1}%
3636    \ifin@
3637      \ifx\bbl@KVP@labels\@nil\else
3638        \bbl@xin@{ map }{ \bbl@KVP@labels\space}%
3639        \ifin@
3640          \def\bbl@tempc{#1}%
3641          \bbl@replace\bbl@tempc{.map}{}%
3642          \in@{,#2,}{,arabic,roman,Roman,alph,Alph,fnsymbol,}%
3643          \bbl@exp{%
3644            \gdef\<bbl@map@\bbl@tempc @\languagename>%
3645              {\ifin@\<#2>\else\\\localecounter{#2}\fi}}%
3646          \bbl@foreach\bbl@list@the{%
3647            \bbl@ifunset{the##1}{}%
3648              {\bbl@exp{\let\\\bbl@tempd\<the##1>}%
3649                \bbl@exp{%
3650                  \\\bbl@sreplace\<the##1>%
3651                    {\<\bbl@tempc>{##1}}{\\\bbl@map@cnt{\bbl@tempc}{##1}}%
3652                  \\\bbl@sreplace\<the##1>%
3653                    {\<\@empty @\bbl@tempc>\<c@##1>}{\\\bbl@map@cnt{\bbl@tempc}{##1}}}%
3654                \expandafter\ifx\csname the##1\endcsname\bbl@tempd\else
3655                  \toks@\expandafter\expandafter\expandafter{%
3656                    \csname the##1\endcsname}%
3657                  \expandafter\xdef\csname the##1\endcsname{{\the\toks@}}%
3658              \fi}}%
3659        \fi
```

```
3660      \fi
3661   %
3662   \else
3663      %
3664      % The following code is still under study. You can test it and make
3665      % suggestions. Eg, enumerate.2 = ([enumi]).([enumii]). It's
3666      % language dependent.
3667      \in@{enumerate.}{#1}%
3668      \ifin@
3669        \def\bbl@tempa{#1}%
3670        \bbl@replace\bbl@tempa{enumerate.}{}%
3671        \def\bbl@toreplace{#2}%
3672        \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
3673        \bbl@replace\bbl@toreplace{[}{\csname the}%
3674        \bbl@replace\bbl@toreplace{]}{\endcsname{}}%
3675        \toks@\expandafter{\bbl@toreplace}%
3676        \bbl@exp{%
3677          \\\bbl@add\<extras\languagename>{%
3678            \\\babel@save\<labelenum\romannumeral\bbl@tempa>%
3679            \def\<labelenum\romannumeral\bbl@tempa>{\the\toks@}}%
3680          \\\bbl@toglobal\<extras\languagename>}%
3681      \fi
3682   \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.

```
3683 \def\bbl@chaptype{chap}
3684 \ifx\@makechapterhead\@undefined
3685   \let\bbl@patchchapter\relax
3686 \else\ifx\thechapter\@undefined
3687   \let\bbl@patchchapter\relax
3688 \else\ifx\ps@headings\@undefined
3689   \let\bbl@patchchapter\relax
3690 \else
3691   \def\bbl@patchchapter{%
3692     \global\let\bbl@patchchapter\relax
3693     \bbl@add\appendix{\def\bbl@chaptype{appx}}% Not harmful, I hope
3694     \bbl@toglobal\appendix
3695     \bbl@sreplace\ps@headings
3696       {\@chapapp\ \thechapter}%
3697       {\bbl@chapterformat}%
3698     \bbl@toglobal\ps@headings
3699     \bbl@sreplace\chaptermark
3700       {\@chapapp\ \thechapter}%
3701       {\bbl@chapterformat}%
3702     \bbl@toglobal\chaptermark
3703     \bbl@sreplace\@makechapterhead
3704       {\@chapapp\space\thechapter}%
3705       {\bbl@chapterformat}%
3706     \bbl@toglobal\@makechapterhead
3707     \gdef\bbl@chapterformat{%
3708       \bbl@ifunset{bbl@\bbl@chaptype fmt@\languagename}%
3709         {\@chapapp\space\thechapter}
3710         {\@nameuse{bbl@\bbl@chaptype fmt@\languagename}}}}}
3711 \fi\fi\fi
3712 \ifx\@part\@undefined
3713   \let\bbl@patchpart\relax
```

```
3714 \else
3715   \def\bbl@patchpart{%
3716     \global\let\bbl@patchpart\relax
3717     \bbl@sreplace\@part
3718       {\partname\nobreakspace\thepart}%
3719       {\bbl@partformat}%
3720     \bbl@toglobal\@part
3721     \gdef\bbl@partformat{%
3722       \bbl@ifunset{bbl@partfmt@\languagename}%
3723         {\partname\nobreakspace\thepart}
3724         {\@nameuse{bbl@partfmt@\languagename}}}}
3725 \fi
```

Date. TODO. Document

```
3726 % Arguments are _not_ protected.
3727 \let\bbl@calendar\@empty
3728 \DeclareRobustCommand\localedate[1][]{\bbl@localedate{#1}}
3729 \def\bbl@localedate#1#2#3#4{%
3730   \begingroup
3731     \ifx\@empty#1\@empty\else
3732       \let\bbl@ld@calendar\@empty
3733       \let\bbl@ld@variant\@empty
3734       \edef\bbl@tempa{\zap@space#1 \@empty}%
3735       \def\bbl@tempb##1=##2\@@{\@namedef{bbl@ld@##1}{##2}}%
3736       \bbl@foreach\bbl@tempa{\bbl@tempb##1\@@}%
3737       \edef\bbl@calendar{%
3738         \bbl@ld@calendar
3739         \ifx\bbl@ld@variant\@empty\else
3740           .\bbl@ld@variant
3741         \fi}%
3742       \bbl@replace\bbl@calendar{gregorian}{}%
3743     \fi
3744     \bbl@cased
3745       {\@nameuse{bbl@date@\languagename @\bbl@calendar}{#2}{#3}{#4}}%
3746   \endgroup}
3747 % eg: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
3748 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
3749   \bbl@trim@def\bbl@tempa{#1.#2}%
3750   \bbl@ifsamestring{\bbl@tempa}{months.wide}%       to savedate
3751     {\bbl@trim@def\bbl@tempa{#3}%
3752     \bbl@trim\toks@{#5}%
3753     \@temptokena\expandafter{\bbl@savedate}%
3754     \bbl@exp{%   Reverse order - in ini last wins
3755       \def\\\bbl@savedate{%
3756         \\\SetString\<month\romannumeral\bbl@tempa#6name>{\the\toks@}%
3757         \the\@temptokena}}%
3758   {\bbl@ifsamestring{\bbl@tempa}{date.long}%       defined now
3759     {\lowercase{\def\bbl@tempb{#6}}%
3760     \bbl@trim@def\bbl@toreplace{#5}%
3761     \bbl@TG@@date
3762     \bbl@ifunset{bbl@date@\languagename @}%
3763       {\global\bbl@csarg\let{date@\languagename @}\bbl@toreplace
3764       % TODO. Move to a better place.
3765       \bbl@exp{%
3766         \gdef\<\languagename date>{\\\protect\<\languagename date >}%
3767         \gdef\<\languagename date >####1####2####3{%
3768           \\\bbl@usedategrouptrue
3769           \<bbl@ensure@\languagename>{%
3770             \\\localedate{####1}{####2}{####3}}}%
```

```
3771              \\\bbl@add\\\bbl@savetoday{%
3772                \\\SetString\\\today{%
3773                  \<\languagename date>%
3774                    {\\\the\year}{\\\the\month}{\\\the\day}}}}}%
3775          {}%
3776        \ifx\bbl@tempb\@empty\else
3777          \global\bbl@csarg\let{date@\languagename @\bbl@tempb}\bbl@toreplace
3778        \fi}%
3779      {}}}
```

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.

```
3780 \let\bbl@calendar\@empty
3781 \newcommand\BabelDateSpace{\nobreakspace}
3782 \newcommand\BabelDateDot{.\@}  % TODO. \let instead of repeating
3783 \newcommand\BabelDated[1]{{\number#1}}
3784 \newcommand\BabelDatedd[1]{{\ifnum#1<10 0\fi\number#1}}
3785 \newcommand\BabelDateM[1]{{\number#1}}
3786 \newcommand\BabelDateMM[1]{{\ifnum#1<10 0\fi\number#1}}
3787 \newcommand\BabelDateMMMM[1]{{%
3788   \csname month\romannumeral#1\bbl@calendar name\endcsname}}%
3789 \newcommand\BabelDatey[1]{{\number#1}}%
3790 \newcommand\BabelDateyy[1]{{%
3791   \ifnum#1<10 0\number#1 %
3792   \else\ifnum#1<100 \number#1 %
3793   \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3794   \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
3795   \else
3796     \bbl@error
3797       {Currently two-digit years are restricted to the\\
3798        range 0-9999.}%
3799       {There is little you can do. Sorry.}%
3800   \fi\fi\fi\fi}}
3801 \newcommand\BabelDateyyyy[1]{{\number#1}} % FIXME - add leading 0
3802 \def\bbl@replace@finish@iii#1{%
3803   \bbl@exp{\def\\#1####1####2####3{\the\toks@}}}
3804 \def\bbl@TG@@date{%
3805   \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace{}}%
3806   \bbl@replace\bbl@toreplace{[.]}{\BabelDateDot{}}%
3807   \bbl@replace\bbl@toreplace{[d]}{\BabelDated{####3}}%
3808   \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{####3}}%
3809   \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{####2}}%
3810   \bbl@replace\bbl@toreplace{[MM]}{\BabelDateMM{####2}}%
3811   \bbl@replace\bbl@toreplace{[MMMM]}{\BabelDateMMMM{####2}}%
3812   \bbl@replace\bbl@toreplace{[y]}{\BabelDatey{####1}}%
3813   \bbl@replace\bbl@toreplace{[yy]}{\BabelDateyy{####1}}%
3814   \bbl@replace\bbl@toreplace{[yyyy]}{\BabelDateyyyy{####1}}%
3815   \bbl@replace\bbl@toreplace{[y|}{\bbl@datecntr[####1|}%
3816   \bbl@replace\bbl@toreplace{[m|}{\bbl@datecntr[####2|}%
3817   \bbl@replace\bbl@toreplace{[d|}{\bbl@datecntr[####3|}%
3818 % Note after \bbl@replace \toks@ contains the resulting string.
3819 % TODO - Using this implicit behavior doesn't seem a good idea.
3820   \bbl@replace@finish@iii\bbl@toreplace}
3821 \def\bbl@datecntr{\expandafter\bbl@xdatecntr\expandafter}
3822 \def\bbl@xdatecntr[#1|#2]{\localenumeral{#2}{#1}}
```

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

```
3823 \def\bbl@provide@lsys#1{%
3824   \bbl@ifunset{bbl@lname@#1}%
3825     {\bbl@ini@basic{#1}}%
3826     {}%
3827   \bbl@csarg\let{lsys@#1}\@empty
3828   \bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}{}%
3829   \bbl@ifunset{bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{}%
3830   \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
3831   \bbl@ifunset{bbl@lname@#1}{}%
3832     {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
3833   \ifcase\bbl@engine\or\or
3834     \bbl@ifunset{bbl@prehc@#1}{}%
3835       {\bbl@exp{\\\bbl@ifblank{\bbl@cs{prehc@#1}}}%
3836         {}%
3837         {\ifx\bbl@xenohyph\@undefined
3838           \let\bbl@xenohyph\bbl@xenohyph@d
3839           \ifx\AtBeginDocument\@notprerr
3840             \expandafter\@secondoftwo  % to execute right now
3841           \fi
3842           \AtBeginDocument{%
3843             \expandafter\bbl@add
3844             \csname selectfont \endcsname{\bbl@xenohyph}%
3845             \expandafter\selectlanguage\expandafter{\languagename}%
3846             \expandafter\bbl@toglobal\csname selectfont \endcsname}%
3847         \fi}}%
3848   \fi
3849   \bbl@csarg\bbl@toglobal{lsys@#1}}
3850 \def\bbl@xenohyph@d{%
3851   \bbl@ifset{bbl@prehc@\languagename}%
3852     {\ifnum\hyphenchar\font=\defaulthyphenchar
3853       \iffontchar\font\bbl@cl{prehc}\relax
3854         \hyphenchar\font\bbl@cl{prehc}\relax
3855       \else\iffontchar\font"200B
3856         \hyphenchar\font"200B
3857       \else
3858         \bbl@warning
3859           {Neither 0 nor ZERO WIDTH SPACE are available\\%
3860            in the current font, and therefore the hyphen\\%
3861            will be printed. Try changing the fontspec's\\%
3862            'HyphenChar' to another value, but be aware\\%
3863            this setting is not safe (see the manual)}%
3864         \hyphenchar\font\defaulthyphenchar
3865       \fi\fi
3866     \fi}%
3867     {\hyphenchar\font\defaulthyphenchar}}
3868   % \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.

```
3869 \def\bbl@ini@basic#1{%
3870   \def\BabelBeforeIni##1##2{%
3871     \begingroup
3872       \bbl@add\bbl@secpost@identification{\closein\bbl@readstream }%
3873       \bbl@read@ini{##1}1%
3874       \endinput            % babel- .tex may contain onlypreamble's
3875     \endgroup}%            boxed, to avoid extra spaces:
```

```
3876    {\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.

```
3877 \def\bbl@setdigits#1#2#3#4#5{%
3878   \bbl@exp{%
3879     \def\<\languagename digits>####1{%         ie, \langdigits
3880       \<bbl@digits@\languagename>####1\\\@nil}%
3881     \let\<bbl@cntr@digits@\languagename>\<\languagename digits>%
3882     \def\<\languagename counter>####1{%        ie, \langcounter
3883       \\\expandafter\<bbl@counter@\languagename>%
3884       \\\csname c@####1\endcsname}%
3885     \def\<bbl@counter@\languagename>####1{% ie, \bbl@counter@lang
3886       \\\expandafter\<bbl@digits@\languagename>%
3887       \\\number####1\\\@nil}}%
3888   \def\bbl@tempa##1##2##3##4##5{%
3889   \bbl@exp{%    Wow, quite a lot of hashes! :-(
3890     \def\<bbl@digits@\languagename>########1{%
3891       \\\ifx########1\\\@nil             % ie, \bbl@digits@lang
3892       \\\else
3893         \\\ifx0########1#1%
3894         \\\else\\\ifx1########1#2%
3895         \\\else\\\ifx2########1#3%
3896         \\\else\\\ifx3########1#4%
3897         \\\else\\\ifx4########1#5%
3898         \\\else\\\ifx5########1##1%
3899         \\\else\\\ifx6########1##2%
3900         \\\else\\\ifx7########1##3%
3901         \\\else\\\ifx8########1##4%
3902         \\\else\\\ifx9########1##5%
3903         \\\else########1%
3904         \\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi
3905         \\\expandafter\<bbl@digits@\languagename>%
3906       \\\fi}}}%
3907   \bbl@tempa}
```

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

```
3908 \def\bbl@buildifcase#1 {% Returns \bbl@tempa, requires \toks@={}
3909   \ifx\\#1%              % \\ before, in case #1 is multiletter
3910     \bbl@exp{%
3911       \def\\\bbl@tempa####1{%
3912         \<ifcase>####1\space\the\toks@\<else>\\\@ctrerr\<fi>}}%
3913   \else
3914     \toks@\expandafter{\the\toks@\or #1}%
3915     \expandafter\bbl@buildifcase
3916   \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).

```
3917 \newcommand\localenumeral[2]{\bbl@cs{cntr@#1@\languagename}{#2}}
3918 \def\bbl@localecntr#1#2{\localenumeral{#2}{#1}}
3919 \newcommand\localecounter[2]{%
3920   \expandafter\bbl@localecntr
```

```
3921    \expandafter{\number\csname c@#2\endcsname}{#1}}
3922 \def\bbl@alphnumeral#1#2{%
3923    \expandafter\bbl@alphnumeral@i\number#2 76543210\@@{#1}}
3924 \def\bbl@alphnumeral@i#1#2#3#4#5#6#7#8\@@#9{%
3925    \ifcase\@car#8\@nil\or   % Currenty <10000, but prepared for bigger
3926      \bbl@alphnumeral@ii{#9}000000#1\or
3927      \bbl@alphnumeral@ii{#9}00000#1#2\or
3928      \bbl@alphnumeral@ii{#9}0000#1#2#3\or
3929      \bbl@alphnumeral@ii{#9}000#1#2#3#4\else
3930      \bbl@alphnum@invalid{>9999}%
3931    \fi}
3932 \def\bbl@alphnumeral@ii#1#2#3#4#5#6#7#8{%
3933    \bbl@ifunset{bbl@cntr@#1.F.\number#5#6#7#8@\languagename}%
3934      {\bbl@cs{cntr@#1.4@\languagename}#5%
3935       \bbl@cs{cntr@#1.3@\languagename}#6%
3936       \bbl@cs{cntr@#1.2@\languagename}#7%
3937       \bbl@cs{cntr@#1.1@\languagename}#8%
3938       \ifnum#6#7#8>\z@ % TODO. An ad hoc rule for Greek. Ugly.
3939         \bbl@ifunset{bbl@cntr@#1.S.321@\languagename}{}%
3940            {\bbl@cs{cntr@#1.S.321@\languagename}}%
3941       \fi}%
3942      {\bbl@cs{cntr@#1.F.\number#5#6#7#8@\languagename}}}
3943 \def\bbl@alphnum@invalid#1{%
3944    \bbl@error{Alphabetic numeral too large (#1)}%
3945      {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.

```
3946 \newcommand\localeinfo[1]{%
3947    \bbl@ifunset{bbl@\csname bbl@info@#1\endcsname @\languagename}%
3948      {\bbl@error{I've found no info for the current locale.\\%
3949                  The corresponding ini file has not been loaded\\%
3950                  Perhaps it doesn't exist}%
3951                 {See the manual for details.}}%
3952      {\bbl@cs{\csname bbl@info@#1\endcsname @\languagename}}}
3953 % \@namedef{bbl@info@name.locale}{lcname}
3954 \@namedef{bbl@info@tag.ini}{lini}
3955 \@namedef{bbl@info@name.english}{elname}
3956 \@namedef{bbl@info@name.opentype}{lname}
3957 \@namedef{bbl@info@tag.bcp47}{lbcp} % TODO
3958 \@namedef{bbl@info@tag.opentype}{lotf}
3959 \@namedef{bbl@info@script.name}{esname}
3960 \@namedef{bbl@info@script.name.opentype}{sname}
3961 \@namedef{bbl@info@script.tag.bcp47}{sbcp}
3962 \@namedef{bbl@info@script.tag.opentype}{sotf}
3963 \let\bbl@ensureinfo\@gobble
3964 \newcommand\BabelEnsureInfo{%
3965    \ifx\InputIfFileExists\@undefined\else
3966      \def\bbl@ensureinfo##1{%
3967        \bbl@ifunset{bbl@lname@##1}{\bbl@ini@basic{##1}}{}}%
3968    \fi
3969    \bbl@foreach\bbl@loaded{{%
3970      \def\languagename{##1}%
3971      \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.

```
3972 \newcommand\getlocaleproperty{%
```

```
3973    \@ifstar\bbl@getproperty@s\bbl@getproperty@x}
3974 \def\bbl@getproperty@s#1#2#3{%
3975    \let#1\relax
3976    \def\bbl@elt##1##2##3{%
3977      \bbl@ifsamestring{##1/##2}{#3}%
3978        {\providecommand#1{##3}%
3979         \def\bbl@elt####1####2####3{}}%
3980        {}}%
3981    \bbl@cs{inidata@#2}}%
3982 \def\bbl@getproperty@x#1#2#3{%
3983    \bbl@getproperty@s{#1}{#2}{#3}%
3984    \ifx#1\relax
3985      \bbl@error
3986        {Unknown key for locale '#2':\\%
3987         #3\\%
3988         \string#1 will be set to \relax}%
3989        {Perhaps you misspelled it.}%
3990    \fi}
3991 \let\bbl@ini@loaded\@empty
3992 \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.

```
3993 \newcommand\babeladjust[1]{%  TODO. Error handling.
3994    \bbl@forkv{#1}{%
3995      \bbl@ifunset{bbl@ADJ@##1@##2}%
3996        {\bbl@cs{ADJ@##1}{##2}}%
3997        {\bbl@cs{ADJ@##1@##2}}}}
3998 %
3999 \def\bbl@adjust@lua#1#2{%
4000    \ifvmode
4001      \ifnum\currentgrouplevel=\z@
4002        \directlua{ Babel.#2 }%
4003        \expandafter\expandafter\expandafter\@gobble
4004      \fi
4005    \fi
4006    {\bbl@error   % The error is gobbled if everything went ok.
4007      {Currently, #1 related features can be adjusted only\\%
4008       in the main vertical list.}%
4009      {Maybe things change in the future, but this is what it is.}}}
4010 \@namedef{bbl@ADJ@bidi.mirroring@on}{%
4011    \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
4012 \@namedef{bbl@ADJ@bidi.mirroring@off}{%
4013    \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
4014 \@namedef{bbl@ADJ@bidi.text@on}{%
4015    \bbl@adjust@lua{bidi}{bidi_enabled=true}}
4016 \@namedef{bbl@ADJ@bidi.text@off}{%
4017    \bbl@adjust@lua{bidi}{bidi_enabled=false}}
4018 \@namedef{bbl@ADJ@bidi.mapdigits@on}{%
4019    \bbl@adjust@lua{bidi}{digits_mapped=true}}
4020 \@namedef{bbl@ADJ@bidi.mapdigits@off}{%
4021    \bbl@adjust@lua{bidi}{digits_mapped=false}}
4022 %
4023 \@namedef{bbl@ADJ@linebreak.sea@on}{%
4024    \bbl@adjust@lua{linebreak}{sea_enabled=true}}
4025 \@namedef{bbl@ADJ@linebreak.sea@off}{%
```

```
4026    \bbl@adjust@lua{linebreak}{sea_enabled=false}}
4027 \@namedef{bbl@ADJ@linebreak.cjk@on}{%
4028    \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
4029 \@namedef{bbl@ADJ@linebreak.cjk@off}{%
4030    \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
4031 %
4032 \def\bbl@adjust@layout#1{%
4033    \ifvmode
4034       #1%
4035       \expandafter\@gobble
4036    \fi
4037    {\bbl@error   % The error is gobbled if everything went ok.
4038       {Currently, layout related features can be adjusted only\\%
4039        in vertical mode.}%
4040       {Maybe things change in the future, but this is what it is.}}}
4041 \@namedef{bbl@ADJ@layout.tabular@on}{%
4042    \bbl@adjust@layout{\let\@tabular\bbl@NL@@tabular}}
4043 \@namedef{bbl@ADJ@layout.tabular@off}{%
4044    \bbl@adjust@layout{\let\@tabular\bbl@OL@@tabular}}
4045 \@namedef{bbl@ADJ@layout.lists@on}{%
4046    \bbl@adjust@layout{\let\list\bbl@NL@list}}
4047 \@namedef{bbl@ADJ@layout.lists@off}{%
4048    \bbl@adjust@layout{\let\list\bbl@OL@list}}
4049 \@namedef{bbl@ADJ@hyphenation.extra@on}{%
4050    \bbl@activateposthyphen}
4051 %
4052 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
4053    \bbl@bcpallowedtrue}
4054 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
4055    \bbl@bcpallowedfalse}
4056 \@namedef{bbl@ADJ@autoload.bcp47.prefix}#1{%
4057    \def\bbl@bcp@prefix{#1}}
4058 \def\bbl@bcp@prefix{bcp47-}
4059 \@namedef{bbl@ADJ@autoload.options}#1{%
4060    \def\bbl@autoload@options{#1}}
4061 \let\bbl@autoload@bcpoptions\@empty
4062 \@namedef{bbl@ADJ@autoload.bcp47.options}#1{%
4063    \def\bbl@autoload@bcpoptions{#1}}
4064 \newif\ifbbl@bcptoname
4065 \@namedef{bbl@ADJ@bcp47.toname@on}{%
4066    \bbl@bcptonametrue
4067    \BabelEnsureInfo}
4068 \@namedef{bbl@ADJ@bcp47.toname@off}{%
4069    \bbl@bcptonamefalse}
4070 % TODO: use babel name, override
4071 %
4072 % As the final task, load the code for lua.
4073 %
4074 \ifx\directlua\@undefined\else
4075    \ifx\bbl@luapatterns\@undefined
4076       \input luababel.def
4077    \fi
4078 \fi
4079 ⟨/core⟩
```

A proxy file for switch.def

```
4080 ⟨∗kernel⟩
4081 \let\bbl@onlyswitch\@empty
4082 \input babel.def
```

156

```
4083 \let\bbl@onlyswitch\@undefined
4084 ⟨/kernel⟩
4085 ⟨∗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.

```
4086 ⟨⟨Make sure ProvidesFile is defined⟩⟩
4087 \ProvidesFile{hyphen.cfg}[⟨⟨date⟩⟩ ⟨⟨version⟩⟩ Babel hyphens]
4088 \xdef\bbl@format{\jobname}
4089 \def\bbl@version{⟨⟨version⟩⟩}
4090 \def\bbl@date{⟨⟨date⟩⟩}
4091 \ifx\AtBeginDocument\@undefined
4092   \def\@empty{}
4093   \let\orig@dump\dump
4094   \def\dump{%
4095     \ifx\@ztryfc\@undefined
4096     \else
4097       \toks0=\expandafter{\@preamblecmds}%
4098       \edef\@preamblecmds{\noexpand\@begindocumenthook\the\toks0}%
4099       \def\@begindocumenthook{}%
4100     \fi
4101     \let\dump\orig@dump\let\orig@dump\@undefined\dump}
4102 \fi
4103 ⟨⟨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.

```
4104 \def\process@line#1#2 #3 #4 {%
4105   \ifx=#1%
4106     \process@synonym{#2}%
4107   \else
4108     \process@language{#1#2}{#3}{#4}%
4109   \fi
4110   \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.

```
4111 \toks@{}
4112 \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.

```
4113 \def\process@synonym#1{%
4114   \ifnum\last@language=\m@ne
4115     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4116   \else
4117     \expandafter\chardef\csname l@#1\endcsname\last@language
4118     \wlog{\string\l@#1=\string\language\the\last@language}%
4119     \expandafter\let\csname #1hyphenmins\expandafter\endcsname
4120       \csname\languagename hyphenmins\endcsname
4121     \let\bbl@elt\relax
4122     \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}{}{}}%
4123   \fi}
```

\process@language   The macro \process@language is used to process a non-empty line from the 'configuration file'. It has three arguments, each delimited by white space. The first argument is the 'name' of a language; the second is the name of the file that contains the patterns. The optional third argument is the name of a file containing hyphenation exceptions.

The first thing to do is call \addlanguage to allocate a pattern register and to make that register 'active'. Then the pattern file is read.

For some hyphenation patterns it is needed to load them with a specific font encoding selected. This can be specified in the file language.dat by adding for instance ':T1' to the name of the language. The macro \bbl@get@enc extracts the font encoding from the language name and stores it in \bbl@hyph@enc. The latter can be used in hyphenation files if you need to set a behavior depending on the given encoding (it is set to empty if no encoding is given).

Pattern files may contain assignments to \lefthyphenmin and \righthyphenmin. TeX does not keep track of these assignments. Therefore we try to detect such assignments and store them in the \⟨lang⟩hyphenmins macro. When no assignments were made we provide a default setting.

Some pattern files contain changes to the \lccode en \uccode arrays. Such changes should remain local to the language; therefore we process the pattern file in a group; the \patterns command acts globally so its effect will be remembered.

Then we globally store the settings of \lefthyphenmin and \righthyphenmin and close the group.

When the hyphenation patterns have been processed we need to see if a file with hyphenation exceptions needs to be read. This is the case when the third argument is not empty and when it does not contain a space token. (Note however there is no need to save hyphenation exceptions into the format.)

\bbl@languages saves a snapshot of the loaded languages in the form \bbl@elt{⟨language-name⟩}{⟨number⟩} {⟨patterns-file⟩}{⟨exceptions-file⟩}. Note the last 2 arguments are empty in 'dialects' defined in language.dat with =. Note also the language name can have encoding info.

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

```
4124 \def\process@language#1#2#3{%
4125   \expandafter\addlanguage\csname l@#1\endcsname
4126   \expandafter\language\csname l@#1\endcsname
4127   \edef\languagename{#1}%
4128   \bbl@hook@everylanguage{#1}%
4129   % > luatex
4130   \bbl@get@enc#1::\@@@
4131   \begingroup
4132     \lefthyphenmin\m@ne
4133     \bbl@hook@loadpatterns{#2}%
4134     % > luatex
```

158

```
4135    \ifnum\lefthyphenmin=\m@ne
4136    \else
4137      \expandafter\xdef\csname #1hyphenmins\endcsname{%
4138         \the\lefthyphenmin\the\righthyphenmin}%
4139    \fi
4140   \endgroup
4141   \def\bbl@tempa{#3}%
4142   \ifx\bbl@tempa\@empty\else
4143     \bbl@hook@loadexceptions{#3}%
4144     % > luatex
4145   \fi
4146   \let\bbl@elt\relax
4147   \edef\bbl@languages{%
4148     \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
4149   \ifnum\the\language=\z@
4150     \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4151       \set@hyphenmins\tw@\thr@@\relax
4152     \else
4153       \expandafter\expandafter\expandafter\set@hyphenmins
4154         \csname #1hyphenmins\endcsname
4155     \fi
4156     \the\toks@
4157     \toks@{}%
4158   \fi}
```

\bbl@get@enc The macro \bbl@get@enc extracts the font encoding from the language name and stores it
\bbl@hyph@enc in \bbl@hyph@enc. It uses delimited arguments to achieve this.

```
4159 \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.

```
4160 \def\bbl@hook@everylanguage#1{}
4161 \def\bbl@hook@loadpatterns#1{\input #1\relax}
4162 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
4163 \def\bbl@hook@loadkernel#1{%
4164   \def\addlanguage{\csname newlanguage\endcsname}%
4165   \def\adddialect##1##2{%
4166     \global\chardef##1##2\relax
4167     \wlog{\string##1 = a dialect from \string\language##2}}%
4168   \def\iflanguage##1{%
4169     \expandafter\ifx\csname l@##1\endcsname\relax
4170       \@nolanerr{##1}%
4171     \else
4172       \ifnum\csname l@##1\endcsname=\language
4173         \expandafter\expandafter\expandafter\@firstoftwo
4174       \else
4175         \expandafter\expandafter\expandafter\@secondoftwo
4176       \fi
4177     \fi}%
4178   \def\providehyphenmins##1##2{%
4179     \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
4180       \@namedef{##1hyphenmins}{##2}%
4181     \fi}%
4182   \def\set@hyphenmins##1##2{%
4183     \lefthyphenmin##1\relax
4184     \righthyphenmin##2\relax}%
4185   \def\selectlanguage{%
```

```
4186    \errhelp{Selecting a language requires a package supporting it}%
4187    \errmessage{Not loaded}}%
4188  \let\foreignlanguage\selectlanguage
4189  \let\otherlanguage\selectlanguage
4190  \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4191  \def\bbl@usehooks##1##2{}% TODO. Temporary!!
4192  \def\setlocale{%
4193    \errhelp{Find an armchair, sit down and wait}%
4194    \errmessage{Not yet available}}%
4195  \let\uselocale\setlocale
4196  \let\locale\setlocale
4197  \let\selectlocale\setlocale
4198  \let\localename\setlocale
4199  \let\textlocale\setlocale
4200  \let\textlanguage\setlocale
4201  \let\languagetext\setlocale}
4202 \begingroup
4203  \def\AddBabelHook#1#2{%
4204    \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
4205      \def\next{\toks1}%
4206    \else
4207      \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname####1}%
4208    \fi
4209    \next}
4210  \ifx\directlua\@undefined
4211    \ifx\XeTeXinputencoding\@undefined\else
4212      \input xebabel.def
4213    \fi
4214  \else
4215    \input luababel.def
4216  \fi
4217  \openin1 = babel-\bbl@format.cfg
4218  \ifeof1
4219  \else
4220    \input babel-\bbl@format.cfg\relax
4221  \fi
4222  \closein1
4223 \endgroup
4224 \bbl@hook@loadkernel{switch.def}
```

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

```
4225 \openin1 = language.dat
```

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

```
4226 \def\languagename{english}%
4227 \ifeof1
4228   \message{I couldn't find the file language.dat,\space
4229           I will try the file hyphen.tex}
4230   \input hyphen.tex\relax
4231   \chardef\l@english\z@
4232 \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.

```
4233   \last@language\m@ne
```

160

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.

```
4234  \loop
4235    \endlinechar\m@ne
4236    \read1 to \bbl@line
4237    \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.

```
4238    \if T\ifeof1F\fi T\relax
4239      \ifx\bbl@line\@empty\else
4240        \edef\bbl@line{\bbl@line\space\space\space}%
4241        \expandafter\process@line\bbl@line\relax
4242      \fi
4243  \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.

```
4244  \begingroup
4245    \def\bbl@elt#1#2#3#4{%
4246      \global\language=#2\relax
4247      \gdef\languagename{#1}%
4248      \def\bbl@elt##1##2##3##4{}}%
4249    \bbl@languages
4250  \endgroup
4251 \fi
4252 \closein1
```

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

```
4253 \if/\the\toks@/\else
4254   \errhelp{language.dat loads no language, only synonyms}
4255   \errmessage{Orphan language synonym}
4256 \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.

```
4257 \let\bbl@line\@undefined
4258 \let\process@line\@undefined
4259 \let\process@synonym\@undefined
4260 \let\process@language\@undefined
4261 \let\bbl@get@enc\@undefined
4262 \let\bbl@hyph@enc\@undefined
4263 \let\bbl@tempa\@undefined
4264 \let\bbl@hook@loadkernel\@undefined
4265 \let\bbl@hook@everylanguage\@undefined
4266 \let\bbl@hook@loadpatterns\@undefined
4267 \let\bbl@hook@loadexceptions\@undefined
4268 ⟨/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].

```
4269 ⟨⟨*More package options⟩⟩ ≡
4270 \chardef\bbl@bidimode\z@
4271 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4272 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
4273 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4274 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4275 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4276 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4277 ⟨⟨/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.

```
4278 ⟨⟨*Font selection⟩⟩ ≡
4279 \bbl@trace{Font handling with fontspec}
4280 \ifx\ExplSyntaxOn\@undefined\else
4281   \ExplSyntaxOn
4282   \catcode`\ =10
4283   \def\bbl@loadfontspec{%
4284     \usepackage{fontspec}%
4285     \expandafter
4286     \def\csname msg~text~>~fontspec/language-not-exist\endcsname##1##2##3##4{%
4287       Font '\l_fontspec_fontname_tl' is using the\\%
4288       default features for language '##1'.\\%
4289       That's usually fine, because many languages\\%
4290       require no specific features, but if the output is\\%
4291       not as expected, consider selecting another font.}
4292     \expandafter
4293     \def\csname msg~text~>~fontspec/no-script\endcsname##1##2##3##4{%
4294       Font '\l_fontspec_fontname_tl' is using the\\%
4295       default features for script '##2'.\\%
4296       That's not always wrong, but if the output is\\%
4297       not as expected, consider selecting another font.}}
4298   \ExplSyntaxOff
4299 \fi
4300 \@onlypreamble\babelfont
4301 \newcommand\babelfont[2][]{%  1=langs/scripts 2=fam
4302   \bbl@foreach{#1}{%
4303     \expandafter\ifx\csname date##1\endcsname\relax
4304     \IfFileExists{babel-##1.tex}%
4305       {\babelprovide{##1}}%
4306       {}%
4307     \fi}%
4308   \edef\bbl@tempa{#1}%
4309   \def\bbl@tempb{#2}%  Used by \bbl@bblfont
4310   \ifx\fontspec\@undefined
4311     \bbl@loadfontspec
4312   \fi
4313   \EnableBabelHook{babel-fontspec}% Just calls \bbl@switchfont
```

```
4314    \bbl@bblfont}
4315 \newcommand\bbl@bblfont[2][]{% 1=features 2=fontname, @font=rm|sf|tt
4316    \bbl@ifunset{\bbl@tempb family}%
4317      {\bbl@providefam{\bbl@tempb}}%
4318      {\bbl@exp{%
4319        \\\bbl@sreplace\<\bbl@tempb family >%
4320          {\@nameuse{\bbl@tempb default}}{\<\bbl@tempb default>}}}%
4321    % For the default font, just in case:
4322    \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
4323    \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
4324      {\bbl@csarg\edef{\bbl@tempb dflt@}{<>{#1}{#2}}% save bbl@rmdflt@
4325       \bbl@exp{%
4326        \let\<bbl@\bbl@tempb dflt@\languagename>\<bbl@\bbl@tempb dflt@>%
4327        \\\bbl@font@set\<bbl@\bbl@tempb dflt@\languagename>%
4328                      \<\bbl@tempb default>\<\bbl@tempb family>}}%
4329      {\bbl@foreach\bbl@tempa{% ie bbl@rmdflt@lang / *scrt
4330        \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:

```
4331 \def\bbl@providefam#1{%
4332    \bbl@exp{%
4333      \\\newcommand\<#1default>{}% Just define it
4334      \\\bbl@add@list\\\bbl@font@fams{#1}%
4335      \\\DeclareRobustCommand\<#1family>{%
4336        \\\not@math@alphabet\<#1family>\relax
4337        \\\fontfamily\<#1default>\\\selectfont}%
4338      \\\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.

```
4339 \def\bbl@nostdfont#1{%
4340    \bbl@ifunset{bbl@WFF@\f@family}%
4341      {\bbl@csarg\gdef{WFF@\f@family}{}%  Flag, to avoid dupl warns
4342       \bbl@infowarn{The current font is not a babel standard family:\\%
4343        #1%
4344        \fontname\font\\%
4345        There is nothing intrinsically wrong with this warning, and\\%
4346        you can ignore it altogether if you do not need these\\%
4347        families. But if they are used in the document, you should be\\%
4348        aware 'babel' will no set Script and Language for them, so\\%
4349        you may consider defining a new family with \string\babelfont.\\%
4350        See the manual for further details about \string\babelfont.\\%
4351        Reported}}
4352      {}}%
4353 \gdef\bbl@switchfont{%
4354    \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
4355    \bbl@exp{%  eg Arabic -> arabic
4356      \lowercase{\edef\\\bbl@tempa{\bbl@cl{sname}}}}%
4357    \bbl@foreach\bbl@font@fams{%
4358      \bbl@ifunset{bbl@##1dflt@\languagename}%    (1) language?
4359        {\bbl@ifunset{bbl@##1dflt@*\bbl@tempa}%    (2) from script?
4360          {\bbl@ifunset{bbl@##1dflt@}%             2=F - (3) from generic?
4361            {}%                                    123=F - nothing!
4362            {\bbl@exp{%                            3=T - from generic
4363              \global\let\<bbl@##1dflt@\languagename>%
4364                        \<bbl@##1dflt@>}}}%
4365          {\bbl@exp{%                              2=T - from script
4366            \global\let\<bbl@##1dflt@\languagename>%
4367                      \<bbl@##1dflt@*\bbl@tempa>}}}%
```

```
4368        {}}%                                    1=T - language, already defined
4369  \def\bbl@tempa{\bbl@nostdfont{}}%
4370  \bbl@foreach\bbl@font@fams{%      don't gather with prev for
4371    \bbl@ifunset{bbl@##1dflt@\languagename}%
4372      {\bbl@cs{famrst@##1}%
4373       \global\bbl@csarg\let{famrst@##1}\relax}%
4374      {\bbl@exp{% order is relevant
4375         \\\bbl@add\\\originalTeX{%
4376           \\\bbl@font@rst{\bbl@cl{##1dflt}}%
4377                        \<##1default>\<##1family>{##1}}%
4378         \\\bbl@font@set\<bbl@##1dflt@\languagename>% the main part!
4379                        \<##1default>\<##1family>}}}%
4380    \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.

```
4381 \ifx\f@family\@undefined\else    % if latex
4382   \ifcase\bbl@engine               % if pdftex
4383     \let\bbl@ckeckstdfonts\relax
4384   \else
4385     \def\bbl@ckeckstdfonts{%
4386       \begingroup
4387         \global\let\bbl@ckeckstdfonts\relax
4388         \let\bbl@tempa\@empty
4389         \bbl@foreach\bbl@font@fams{%
4390           \bbl@ifunset{bbl@##1dflt@}%
4391             {\@nameuse{##1family}%
4392              \bbl@csarg\gdef{WFF@\f@family}{}% Flag
4393              \bbl@exp{\\\bbl@add\\\bbl@tempa{* \<##1family>= \f@family\\\\%
4394                 \space\space\fontname\font\\\\}}%
4395              \bbl@csarg\xdef{##1dflt@}{\f@family}%
4396              \expandafter\xdef\csname ##1default\endcsname{\f@family}}%
4397             {}}%
4398         \ifx\bbl@tempa\@empty\else
4399           \bbl@infowarn{The following font families will use the default\\%
4400             settings for all or some languages:\\%
4401             \bbl@tempa
4402             There is nothing intrinsically wrong with it, but\\%
4403             'babel' will no set Script and Language, which could\\%
4404              be relevant in some languages. If your document uses\\%
4405              these families, consider redefining them with \string\babelfont.\\%
4406             Reported}%
4407         \fi
4408       \endgroup}
4409   \fi
4410 \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.

```
4411 \def\bbl@font@set#1#2#3{% eg \bbl@rmdflt@lang \rmdefault \rmfamily
4412   \bbl@xin@{<>}{#1}%
4413   \ifin@
4414     \bbl@exp{\\\bbl@fontspec@set\\#1\expandafter\@gobbletwo#1\\#3}%
4415   \fi
4416   \bbl@exp{%
4417     \def\\#2{#1}%          eg, \rmdefault{\bbl@rmdflt@lang}
```

164

```
4418     \\\bbl@ifsamestring{#2}{\f@family}{\\#3\let\\\bbl@tempa\relax}{}}}
4419%    TODO - next should be global?, but even local does its job. I'm
4420%    still not sure -- must investigate:
4421\def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
4422   \let\bbl@tempe\bbl@mapselect
4423   \let\bbl@mapselect\relax
4424   \let\bbl@temp@fam#4%        eg, '\rmfamily', to be restored below
4425   \let#4\@empty       %        Make sure \renewfontfamily is valid
4426   \bbl@exp{%
4427     \let\\\bbl@temp@pfam\<\bbl@stripslash#4\space>% eg, '\rmfamily '
4428     \<keys_if_exist:nnF>{fontspec-opentype}{Script/\bbl@cl{sname}}%
4429       {\\\newfontscript{\bbl@cl{sname}}{\bbl@cl{sotf}}}%
4430     \<keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@cl{lname}}%
4431       {\\\newfontlanguage{\bbl@cl{lname}}{\bbl@cl{lotf}}}%
4432     \\\renewfontfamily\\#4%
4433       [\bbl@cs{lsys@languagename},#2]}{#3}% ie \bbl@exp{..}{#3}
4434   \begingroup
4435     #4%
4436     \xdef#1{\f@family}%     eg, \bbl@rmdflt@lang{FreeSerif(0)}
4437   \endgroup
4438   \let#4\bbl@temp@fam
4439   \bbl@exp{\let\<\bbl@stripslash#4\space>}\bbl@temp@pfam
4440   \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.

```
4441\def\bbl@font@rst#1#2#3#4{%
4442   \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.

```
4443\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 :-).

```
4444\newcommand\babelFSstore[2][]{%
4445   \bbl@ifblank{#1}%
4446     {\bbl@csarg\def{sname@#2}{Latin}}%
4447     {\bbl@csarg\def{sname@#2}{#1}}%
4448   \bbl@provide@dirs{#2}%
4449   \bbl@csarg\ifnum{wdir@#2}>\z@
4450     \let\bbl@beforeforeign\leavevmode
4451     \EnableBabelHook{babel-bidi}%
4452   \fi
4453   \bbl@foreach{#2}{%
4454     \bbl@FSstore{##1}{rm}\rmdefault\bbl@save@rmdefault
4455     \bbl@FSstore{##1}{sf}\sfdefault\bbl@save@sfdefault
4456     \bbl@FSstore{##1}{tt}\ttdefault\bbl@save@ttdefault}}
4457\def\bbl@FSstore#1#2#3#4{%
4458   \bbl@csarg\edef{#2default#1}{#3}%
4459   \expandafter\addto\csname extras#1\endcsname{%
4460     \let#4#3%
4461     \ifx#3\f@family
4462       \edef#3{\csname bbl@#2default#1\endcsname}%
4463       \fontfamily{#3}\selectfont
4464     \else
4465       \edef#3{\csname bbl@#2default#1\endcsname}%
4466     \fi}%
```

```
4467  \expandafter\addto\csname noextras#1\endcsname{%
4468    \ifx#3\f@family
4469      \fontfamily{#4}\selectfont
4470    \fi
4471    \let#3#4}}
4472 \let\bbl@langfeatures\@empty
4473 \def\babelFSfeatures{% make sure \fontspec is redefined once
4474   \let\bbl@ori@fontspec\fontspec
4475   \renewcommand\fontspec[1][]{%
4476     \bbl@ori@fontspec[\bbl@langfeatures##1]}
4477   \let\babelFSfeatures\bbl@FSfeatures
4478   \babelFSfeatures}
4479 \def\bbl@FSfeatures#1#2{%
4480   \expandafter\addto\csname extras#1\endcsname{%
4481     \babel@save\bbl@langfeatures
4482     \edef\bbl@langfeatures{#2,}}}
4483 ⟨⟨/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.

```
4484 ⟨⟨*Footnote changes⟩⟩ ≡
4485 \bbl@trace{Bidi footnotes}
4486 \ifnum\bbl@bidimode>\z@
4487   \def\bbl@footnote#1#2#3{%
4488     \@ifnextchar[%
4489       {\bbl@footnote@o{#1}{#2}{#3}}%
4490       {\bbl@footnote@x{#1}{#2}{#3}}}
4491   \long\def\bbl@footnote@x#1#2#3#4{%
4492     \bgroup
4493       \select@language@x{\bbl@main@language}%
4494       \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
4495     \egroup}
4496   \long\def\bbl@footnote@o#1#2#3[#4]#5{%
4497     \bgroup
4498       \select@language@x{\bbl@main@language}%
4499       \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
4500     \egroup}
4501   \def\bbl@footnotetext#1#2#3{%
4502     \@ifnextchar[%
4503       {\bbl@footnotetext@o{#1}{#2}{#3}}%
4504       {\bbl@footnotetext@x{#1}{#2}{#3}}}
4505   \long\def\bbl@footnotetext@x#1#2#3#4{%
4506     \bgroup
4507       \select@language@x{\bbl@main@language}%
4508       \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
4509     \egroup}
4510   \long\def\bbl@footnotetext@o#1#2#3[#4]#5{%
4511     \bgroup
4512       \select@language@x{\bbl@main@language}%
4513       \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
4514     \egroup}
4515   \def\BabelFootnote#1#2#3#4{%
4516     \ifx\bbl@fn@footnote\@undefined
```

```
4517        \let\bbl@fn@footnote\footnote
4518      \fi
4519      \ifx\bbl@fn@footnotetext\@undefined
4520        \let\bbl@fn@footnotetext\footnotetext
4521      \fi
4522      \bbl@ifblank{#2}%
4523        {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
4524         \@namedef{\bbl@stripslash#1text}%
4525           {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
4526        {\def#1{\bbl@exp{\\\bbl@footnote{\\\foreignlanguage{#2}}}{#3}{#4}}%
4527         \@namedef{\bbl@stripslash#1text}%
4528           {\bbl@exp{\\\bbl@footnotetext{\\\foreignlanguage{#2}}}{#3}{#4}}}}
4529  \fi
4530  ⟨⟨/Footnote changes⟩⟩
```

Now, the code.

```
4531  ⟨*xetex⟩
4532  \def\BabelStringsDefault{unicode}
4533  \let\xebbl@stop\relax
4534  \AddBabelHook{xetex}{encodedcommands}{%
4535      \def\bbl@tempa{#1}%
4536      \ifx\bbl@tempa\@empty
4537        \XeTeXinputencoding"bytes"%
4538      \else
4539        \XeTeXinputencoding"#1"%
4540      \fi
4541      \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4542  \AddBabelHook{xetex}{stopcommands}{%
4543      \xebbl@stop
4544      \let\xebbl@stop\relax}
4545  \def\bbl@intraspace#1 #2 #3\@@{%
4546      \bbl@csarg\gdef{xeisp@\languagename}%
4547        {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4548  \def\bbl@intrapenalty#1\@@{%
4549      \bbl@csarg\gdef{xeipn@\languagename}%
4550        {\XeTeXlinebreakpenalty #1\relax}}
4551  \def\bbl@provide@intraspace{%
4552      \bbl@xin@{\bbl@cl{lnbrk}}{s}%
4553      \ifin@\else\bbl@xin@{\bbl@cl{lnbrk}}{c}\fi
4554      \ifin@
4555        \bbl@ifunset{bbl@intsp@\languagename}{}%
4556          {\expandafter\ifx\csname bbl@intsp@\languagename\endcsname\@empty\else
4557            \ifx\bbl@KVP@intraspace\@nil
4558              \bbl@exp{%
4559                \\\bbl@intraspace\bbl@cl{intsp}\\\@@}%
4560            \fi
4561            \ifx\bbl@KVP@intrapenalty\@nil
4562              \bbl@intrapenalty0\@@
4563            \fi
4564          \fi
4565          \ifx\bbl@KVP@intraspace\@nil\else % We may override the ini
4566            \expandafter\bbl@intraspace\bbl@KVP@intraspace\@@
4567          \fi
4568          \ifx\bbl@KVP@intrapenalty\@nil\else
4569            \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
4570          \fi
4571          \bbl@exp{%
4572            \\\bbl@add\<extras\languagename>{%
4573              \XeTeXlinebreaklocale "\bbl@cl{lbcp}"%
```

```
4574          \<bbl@xeisp@\languagename>%
4575          \<bbl@xeipn@\languagename>}%
4576        \\\bbl@toglobal\<extras\languagename>%
4577        \\\bbl@add\<noextras\languagename>{%
4578          \XeTeXlinebreaklocale "en"}%
4579        \\\bbl@toglobal\<noextras\languagename>}%
4580      \ifx\bbl@ispacesize\@undefined
4581        \gdef\bbl@ispacesize{\bbl@cl{xeisp}}%
4582        \ifx\AtBeginDocument\@notprerr
4583          \expandafter\@secondoftwo  % to execute right now
4584        \fi
4585        \AtBeginDocument{%
4586          \expandafter\bbl@add
4587          \csname selectfont \endcsname{\bbl@ispacesize}%
4588          \expandafter\bbl@toglobal\csname selectfont \endcsname}%
4589      \fi}%
4590  \fi}
4591 \ifx\DisableBabelHook\@undefined\endinput\fi
4592 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4593 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
4594 \DisableBabelHook{babel-fontspec}
4595 ⟨⟨Font selection⟩⟩
4596 \input txtbabel.def
4597 ⟨/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.

```
4598 ⟨*texxet⟩
4599 \providecommand\bbl@provide@intraspace{}
4600 \bbl@trace{Redefinitions for bidi layout}
4601 \def\bbl@sspre@caption{%
4602   \bbl@exp{\everyhbox{\\\bbl@textdir\bbl@cs{wdir@\bbl@main@language}}}}
4603 \ifx\bbl@opt@layout\@nnil\endinput\fi  % No layout
4604 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
4605 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
4606 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
4607   \def\@hangfrom#1{%
4608     \setbox\@tempboxa\hbox{{#1}}%
4609     \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
4610     \noindent\box\@tempboxa}
4611   \def\raggedright{%
4612     \let\\\@centercr
4613     \bbl@startskip\z@skip
4614     \@rightskip\@flushglue
4615     \bbl@endskip\@rightskip
4616     \parindent\z@
4617     \parfillskip\bbl@startskip}
4618   \def\raggedleft{%
4619     \let\\\@centercr
```

```
4620       \bbl@startskip\@flushglue
4621       \bbl@endskip\z@skip
4622       \parindent\z@
4623       \parfillskip\bbl@endskip}
4624 \fi
4625 \IfBabelLayout{lists}
4626   {\bbl@sreplace\list
4627       {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
4628    \def\bbl@listleftmargin{%
4629       \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
4630    \ifcase\bbl@engine
4631       \def\labelenumii{)\theenumii(}% pdftex doesn't reverse ()
4632       \def\p@enumiii{\p@enumii)\theenumii(}%
4633    \fi
4634    \bbl@sreplace\@verbatim
4635       {\leftskip\@totalleftmargin}%
4636       {\bbl@startskip\textwidth
4637        \advance\bbl@startskip-\linewidth}%
4638    \bbl@sreplace\@verbatim
4639       {\rightskip\z@skip}%
4640       {\bbl@endskip\z@skip}}%
4641    {}
4642 \IfBabelLayout{contents}
4643   {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
4644    \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
4645    {}
4646 \IfBabelLayout{columns}
4647   {\bbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbl@outputhbox}%
4648    \def\bbl@outputhbox#1{%
4649       \hb@xt@\textwidth{%
4650          \hskip\columnwidth
4651          \hfil
4652          {\normalcolor\vrule \@width\columnseprule}%
4653          \hfil
4654          \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
4655          \hskip-\textwidth
4656          \hb@xt@\columnwidth{\box\@outputbox \hss}%
4657          \hskip\columnsep
4658          \hskip\columnwidth}}}%
4659    {}
4660 ⟨⟨Footnote changes⟩⟩
4661 \IfBabelLayout{footnotes}%
4662   {\BabelFootnote\footnote\languagename{}{}%
4663    \BabelFootnote\localfootnote\languagename{}{}%
4664    \BabelFootnote\mainfootnote{}{}{}}
4665    {}
```

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.

```
4666 \IfBabelLayout{counters}%
4667   {\let\bbl@latinarabic=\@arabic
4668    \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
4669    \let\bbl@asciiroman=\@roman
4670    \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciiroman#1}}}%
4671    \let\bbl@asciiRoman=\@Roman
4672    \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}}{}
4673 ⟨/texxet⟩
```

169

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`).

```
4674 ⟨*luatex⟩
4675 \ifx\AddBabelHook\@undefined % When plain.def, babel.sty starts
4676 \bbl@trace{Read language.dat}
4677 \ifx\bbl@readstream\@undefined
4678   \csname newread\endcsname\bbl@readstream
4679 \fi
4680 \begingroup
4681   \toks@{}
4682   \count@\z@ % 0=start, 1=0th, 2=normal
4683   \def\bbl@process@line#1#2 #3 #4 {%
4684     \ifx=#1%
4685       \bbl@process@synonym{#2}%
4686     \else
4687       \bbl@process@language{#1#2}{#3}{#4}%
4688     \fi
4689     \ignorespaces}
4690   \def\bbl@manylang{%
4691     \ifnum\bbl@last>\@ne
4692       \bbl@info{Non-standard hyphenation setup}%
4693     \fi
```

```
4694        \let\bbl@manylang\relax}
4695   \def\bbl@process@language#1#2#3{%
4696       \ifcase\count@
4697          \@ifundefined{zth@#1}{\count@\tw@}{\count@\@ne}%
4698       \or
4699          \count@\tw@
4700       \fi
4701       \ifnum\count@=\tw@
4702          \expandafter\addlanguage\csname l@#1\endcsname
4703          \language\allocationnumber
4704          \chardef\bbl@last\allocationnumber
4705          \bbl@manylang
4706          \let\bbl@elt\relax
4707          \xdef\bbl@languages{%
4708             \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
4709       \fi
4710       \the\toks@
4711       \toks@{}}
4712   \def\bbl@process@synonym@aux#1#2{%
4713       \global\expandafter\chardef\csname l@#1\endcsname#2\relax
4714       \let\bbl@elt\relax
4715       \xdef\bbl@languages{%
4716          \bbl@languages\bbl@elt{#1}{#2}{}{}}}%
4717   \def\bbl@process@synonym#1{%
4718       \ifcase\count@
4719          \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
4720       \or
4721          \@ifundefined{zth@#1}{\bbl@process@synonym@aux{#1}{0}}{}%
4722       \else
4723          \bbl@process@synonym@aux{#1}{\the\bbl@last}%
4724       \fi}
4725   \ifx\bbl@languages\@undefined % Just a (sensible?) guess
4726       \chardef\l@english\z@
4727       \chardef\l@USenglish\z@
4728       \chardef\bbl@last\z@
4729       \global\@namedef{bbl@hyphendata@0}{{hyphen.tex}{}}
4730       \gdef\bbl@languages{%
4731          \bbl@elt{english}{0}{hyphen.tex}{}%
4732          \bbl@elt{USenglish}{0}{}{}}
4733   \else
4734       \global\let\bbl@languages@format\bbl@languages
4735       \def\bbl@elt#1#2#3#4{% Remove all except language 0
4736          \ifnum#2>\z@\else
4737             \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
4738          \fi}%
4739       \xdef\bbl@languages{\bbl@languages}%
4740   \fi
4741   \def\bbl@elt#1#2#3#4{\@namedef{zth@#1}{}} % Define flags
4742   \bbl@languages
4743   \openin\bbl@readstream=language.dat
4744   \ifeof\bbl@readstream
4745       \bbl@warning{I couldn't find language.dat. No additional\\%
4746                    patterns loaded. Reported}%
4747   \else
4748       \loop
4749          \endlinechar\m@ne
4750          \read\bbl@readstream to \bbl@line
4751          \endlinechar`\^^M
4752          \if T\ifeof\bbl@readstream F\fi T\relax
```

```
4753      \ifx\bbl@line\@empty\else
4754        \edef\bbl@line{\bbl@line\space\space\space}%
4755        \expandafter\bbl@process@line\bbl@line\relax
4756      \fi
4757    \repeat
4758  \fi
4759 \endgroup
4760 \bbl@trace{Macros for reading patterns files}
4761 \def\bbl@get@enc#1:#2:#3\@@@{\def\bbl@hyph@enc{#2}}
4762 \ifx\babelcatcodetablenum\@undefined
4763   \ifx\newcatcodetable\@undefined
4764     \def\babelcatcodetablenum{5211}
4765     \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4766   \else
4767     \newcatcodetable\babelcatcodetablenum
4768     \newcatcodetable\bbl@pattcodes
4769   \fi
4770 \else
4771   \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4772 \fi
4773 \def\bbl@luapatterns#1#2{%
4774   \bbl@get@enc#1::\@@@
4775   \setbox\z@\hbox\bgroup
4776     \begingroup
4777       \savecatcodetable\babelcatcodetablenum\relax
4778       \initcatcodetable\bbl@pattcodes\relax
4779       \catcodetable\bbl@pattcodes\relax
4780         \catcode`\#=6  \catcode`\$=3 \catcode`\&=4 \catcode`\^=7
4781         \catcode`\_=8  \catcode`\{=1 \catcode`\}=2 \catcode`\~=13
4782         \catcode`\@=11 \catcode`\^^I=10 \catcode`\^^J=12
4783         \catcode`\<=12 \catcode`\>=12 \catcode`\*=12 \catcode`\.=12
4784         \catcode`\-=12 \catcode`\/=12 \catcode`\[=12 \catcode`\]=12
4785         \catcode`\`=12 \catcode`\'=12 \catcode`\"=12
4786         \input #1\relax
4787       \catcodetable\babelcatcodetablenum\relax
4788     \endgroup
4789     \def\bbl@tempa{#2}%
4790     \ifx\bbl@tempa\@empty\else
4791       \input #2\relax
4792     \fi
4793   \egroup}%
4794 \def\bbl@patterns@lua#1{%
4795   \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
4796     \csname l@#1\endcsname
4797     \edef\bbl@tempa{#1}%
4798   \else
4799     \csname l@#1:\f@encoding\endcsname
4800     \edef\bbl@tempa{#1:\f@encoding}%
4801   \fi\relax
4802   \@namedef{lu@texhyphen@loaded@\the\language}{}% Temp
4803   \@ifundefined{bbl@hyphendata@\the\language}%
4804     {\def\bbl@elt##1##2##3##4{%
4805        \ifnum##2=\csname l@\bbl@tempa\endcsname % #2=spanish, dutch:OT1...
4806          \def\bbl@tempb{##3}%
4807          \ifx\bbl@tempb\@empty\else % if not a synonymous
4808            \def\bbl@tempc{{##3}{##4}}%
4809          \fi
4810          \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
4811        \fi}%
```

172

```
4812        \bbl@languages
4813        \@ifundefined{bbl@hyphendata@\the\language}%
4814          {\bbl@info{No hyphenation patterns were set for\\%
4815                    language '\bbl@tempa'. Reported}}%
4816          {\expandafter\expandafter\expandafter\bbl@luapatterns
4817             \csname bbl@hyphendata@\the\language\endcsname}}{}}
4818 \endinput\fi
4819   % Here ends \ifx\AddBabelHook\@undefined
4820   % A few lines are only read by hyphen.cfg
4821 \ifx\DisableBabelHook\@undefined
4822   \AddBabelHook{luatex}{everylanguage}{%
4823     \def\process@language##1##2##3{%
4824       \def\process@line####1####2 ####3 ####4 {}}}
4825   \AddBabelHook{luatex}{loadpatterns}{%
4826       \input #1\relax
4827       \expandafter\gdef\csname bbl@hyphendata@\the\language\endcsname
4828         {{#1}{}}}
4829   \AddBabelHook{luatex}{loadexceptions}{%
4830       \input #1\relax
4831       \def\bbl@tempb##1##2{{##1}{#1}}%
4832       \expandafter\xdef\csname bbl@hyphendata@\the\language\endcsname
4833         {\expandafter\expandafter\expandafter\bbl@tempb
4834          \csname bbl@hyphendata@\the\language\endcsname}}
4835 \endinput\fi
4836   % Here stops reading code for hyphen.cfg
4837   % The following is read the 2nd time it's loaded
4838 \begingroup
4839 \catcode`\%=12
4840 \catcode`\'=12
4841 \catcode`\"=12
4842 \catcode`\:=12
4843 \directlua{
4844   Babel = Babel or {}
4845   function Babel.bytes(line)
4846     return line:gsub("(.)",
4847       function (chr) return unicode.utf8.char(string.byte(chr)) end)
4848   end
4849   function Babel.begin_process_input()
4850     if luatexbase and luatexbase.add_to_callback then
4851       luatexbase.add_to_callback('process_input_buffer',
4852                                  Babel.bytes,'Babel.bytes')
4853     else
4854       Babel.callback = callback.find('process_input_buffer')
4855       callback.register('process_input_buffer',Babel.bytes)
4856     end
4857   end
4858   function Babel.end_process_input ()
4859     if luatexbase and luatexbase.remove_from_callback then
4860       luatexbase.remove_from_callback('process_input_buffer','Babel.bytes')
4861     else
4862       callback.register('process_input_buffer',Babel.callback)
4863     end
4864   end
4865   function Babel.addpatterns(pp, lg)
4866     local lg = lang.new(lg)
4867     local pats = lang.patterns(lg) or ''
4868     lang.clear_patterns(lg)
4869     for p in pp:gmatch('[^%s]+') do
4870       ss = ''
```

```
4871        for i in string.utfcharacters(p:gsub('%d', '')) do
4872          ss = ss .. '%d?' .. i
4873        end
4874        ss = ss:gsub('^%%d%?%.', '%%.') .. '%d?'
4875        ss = ss:gsub('%.%%d%?$', '%%.')
4876        pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
4877        if n == 0 then
4878          tex.sprint(
4879            [[\string\csname\space bbl@info\endcsname{New pattern: ]]
4880            .. p .. [[}]])
4881          pats = pats .. ' ' .. p
4882        else
4883          tex.sprint(
4884            [[\string\csname\space bbl@info\endcsname{Renew pattern: ]]
4885            .. p .. [[}]])
4886        end
4887      end
4888      lang.patterns(lg, pats)
4889  end
4890 }
4891 \endgroup
4892 \ifx\newattribute\@undefined\else
4893   \newattribute\bbl@attr@locale
4894   \directlua{ Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale'}
4895   \AddBabelHook{luatex}{beforeextras}{%
4896     \setattribute\bbl@attr@locale\localeid}
4897 \fi
4898 \def\BabelStringsDefault{unicode}
4899 \let\luabbl@stop\relax
4900 \AddBabelHook{luatex}{encodedcommands}{%
4901   \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
4902   \ifx\bbl@tempa\bbl@tempb\else
4903     \directlua{Babel.begin_process_input()}%
4904     \def\luabbl@stop{%
4905       \directlua{Babel.end_process_input()}}%
4906   \fi}%
4907 \AddBabelHook{luatex}{stopcommands}{%
4908   \luabbl@stop
4909   \let\luabbl@stop\relax}
4910 \AddBabelHook{luatex}{patterns}{%
4911   \@ifundefined{bbl@hyphendata@\the\language}%
4912     {\def\bbl@elt##1##2##3##4{%
4913        \ifnum##2=\csname l@##2\endcsname % #2=spanish, dutch:OT1...
4914          \def\bbl@tempb{##3}%
4915          \ifx\bbl@tempb\@empty\else % if not a synonymous
4916            \def\bbl@tempc{{##3}{##4}}%
4917          \fi
4918          \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
4919        \fi}%
4920      \bbl@languages
4921      \@ifundefined{bbl@hyphendata@\the\language}%
4922        {\bbl@info{No hyphenation patterns were set for\\%
4923                  language '#2'. Reported}}%
4924        {\expandafter\expandafter\expandafter\bbl@luapatterns
4925          \csname bbl@hyphendata@\the\language\endcsname}}{}%
4926   \@ifundefined{bbl@patterns@}{}{%
4927     \begingroup
4928       \bbl@xin@{,\number\language,}{,\bbl@pttnlist}%
4929       \ifin@\else
```

174

```
4930          \ifx\bbl@patterns@\@empty\else
4931            \directlua{ Babel.addpatterns(
4932              [[\bbl@patterns@]], \number\language) }%
4933          \fi
4934          \@ifundefined{bbl@patterns@#1}%
4935            \@empty
4936            {\directlua{ Babel.addpatterns(
4937                [[\space\csname bbl@patterns@#1\endcsname]],
4938                \number\language) }}%
4939          \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
4940        \fi
4941      \endgroup}%
4942  \bbl@exp{%
4943    \bbl@ifunset{bbl@prehc@\languagename}{}%
4944      {\\\bbl@ifblank{\bbl@cs{prehc@\languagename}}{}%
4945        {\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.

```
4946 \@onlypreamble\babelpatterns
4947 \AtEndOfPackage{%
4948   \newcommand\babelpatterns[2][\@empty]{%
4949     \ifx\bbl@patterns@\relax
4950       \let\bbl@patterns@\@empty
4951     \fi
4952     \ifx\bbl@pttnlist\@empty\else
4953       \bbl@warning{%
4954         You must not intermingle \string\selectlanguage\space and\\%
4955         \string\babelpatterns\space or some patterns will not\\%
4956         be taken into account. Reported}%
4957     \fi
4958     \ifx\@empty#1%
4959       \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
4960     \else
4961       \edef\bbl@tempb{\zap@space#1 \@empty}%
4962       \bbl@for\bbl@tempa\bbl@tempb{%
4963         \bbl@fixname\bbl@tempa
4964         \bbl@iflanguage\bbl@tempa{%
4965           \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
4966             \@ifundefined{bbl@patterns@\bbl@tempa}%
4967               \@empty
4968               {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
4969             #2}}}%
4970     \fi}}
```

13.4   Southeast Asian scripts

First, some general code for line breaking, used by \babelposthyphenation.
In progress. 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.
For the moment, only 3 SA languages are activated by default (see Unicode UAX 14).

```
4971 \directlua{
4972   Babel = Babel or {}
4973   Babel.linebreaking = Babel.linebreaking or {}
4974   Babel.linebreaking.before = {}
```

```
4975  Babel.linebreaking.after = {}
4976  Babel.locale = {} % Free to use, indexed with \localeid
4977  function Babel.linebreaking.add_before(func)
4978    tex.print([[\noexpand\csname bbl@luahyphenate\endcsname]])
4979    table.insert(Babel.linebreaking.before , func)
4980  end
4981  function Babel.linebreaking.add_after(func)
4982    tex.print([[\noexpand\csname bbl@luahyphenate\endcsname]])
4983    table.insert(Babel.linebreaking.after, func)
4984  end
4985 }
4986 \def\bbl@intraspace#1 #2 #3\@@{%
4987  \directlua{
4988    Babel = Babel or {}
4989    Babel.intraspaces = Babel.intraspaces or {}
4990    Babel.intraspaces['\csname bbl@sbcp@\languagename\endcsname'] = %
4991      {b = #1, p = #2, m = #3}
4992    Babel.locale_props[\the\localeid].intraspace = %
4993      {b = #1, p = #2, m = #3}
4994  }}
4995 \def\bbl@intrapenalty#1\@@{%
4996  \directlua{
4997    Babel = Babel or {}
4998    Babel.intrapenalties = Babel.intrapenalties or {}
4999    Babel.intrapenalties['\csname bbl@sbcp@\languagename\endcsname'] = #1
5000    Babel.locale_props[\the\localeid].intrapenalty = #1
5001  }}
5002 \begingroup
5003 \catcode`\%=12
5004 \catcode`\^=14
5005 \catcode`\'=12
5006 \catcode`\~=12
5007 \gdef\bbl@seaintraspace{^
5008  \let\bbl@seaintraspace\relax
5009  \directlua{
5010    Babel = Babel or {}
5011    Babel.sea_enabled = true
5012    Babel.sea_ranges = Babel.sea_ranges or {}
5013    function Babel.set_chranges (script, chrng)
5014      local c = 0
5015      for s, e in string.gmatch(chrng..' ', '(.-)%.%.(.-)%s') do
5016        Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
5017        c = c + 1
5018      end
5019    end
5020    function Babel.sea_disc_to_space (head)
5021      local sea_ranges = Babel.sea_ranges
5022      local last_char = nil
5023      local quad = 655360      ^^ 10 pt = 655360 = 10 * 65536
5024      for item in node.traverse(head) do
5025        local i = item.id
5026        if i == node.id'glyph' then
5027          last_char = item
5028        elseif i == 7 and item.subtype == 3 and last_char
5029            and last_char.char > 0x0C99 then
5030          quad = font.getfont(last_char.font).size
5031          for lg, rg in pairs(sea_ranges) do
5032            if last_char.char > rg[1] and last_char.char < rg[2] then
5033              lg = lg:sub(1, 4)  ^^ Remove trailing number of, eg, Cyrl1
```

176

```
5034              local intraspace = Babel.intraspaces[lg]
5035              local intrapenalty = Babel.intrapenalties[lg]
5036              local n
5037              if intrapenalty ~= 0 then
5038                n = node.new(14, 0)      ^^ penalty
5039                n.penalty = intrapenalty
5040                node.insert_before(head, item, n)
5041              end
5042              n = node.new(12, 13)      ^^ (glue, spaceskip)
5043              node.setglue(n, intraspace.b * quad,
5044                             intraspace.p * quad,
5045                             intraspace.m * quad)
5046              node.insert_before(head, item, n)
5047              node.remove(head, item)
5048            end
5049          end
5050        end
5051      end
5052    end
5053  }^^
5054  \bbl@luahyphenate}
5055 \catcode`\%=14
5056 \gdef\bbl@cjkintraspace{%
5057  \let\bbl@cjkintraspace\relax
5058  \directlua{
5059    Babel = Babel or {}
5060    require'babel-data-cjk.lua'
5061    Babel.cjk_enabled = true
5062    function Babel.cjk_linebreak(head)
5063      local GLYPH = node.id'glyph'
5064      local last_char = nil
5065      local quad = 655360      % 10 pt = 655360 = 10 * 65536
5066      local last_class = nil
5067      local last_lang = nil
5068
5069      for item in node.traverse(head) do
5070        if item.id == GLYPH then
5071
5072          local lang = item.lang
5073
5074          local LOCALE = node.get_attribute(item,
5075              luatexbase.registernumber'bbl@attr@locale')
5076          local props = Babel.locale_props[LOCALE]
5077
5078          local class = Babel.cjk_class[item.char].c
5079
5080          if class == 'cp' then class = 'cl' end % )] as CL
5081          if class == 'id' then class = 'I' end
5082
5083          local br = 0
5084          if class and last_class and Babel.cjk_breaks[last_class][class] then
5085            br = Babel.cjk_breaks[last_class][class]
5086          end
5087
5088          if br == 1 and props.linebreak == 'c' and
5089              lang ~= \the\l@nohyphenation\space and
5090              last_lang ~= \the\l@nohyphenation then
5091            local intrapenalty = props.intrapenalty
5092            if intrapenalty ~= 0 then
```

```
5093            local n = node.new(14, 0)      % penalty
5094            n.penalty = intrapenalty
5095            node.insert_before(head, item, n)
5096          end
5097          local intraspace = props.intraspace
5098          local n = node.new(12, 13)      % (glue, spaceskip)
5099          node.setglue(n, intraspace.b * quad,
5100                         intraspace.p * quad,
5101                         intraspace.m * quad)
5102          node.insert_before(head, item, n)
5103        end
5104
5105        if font.getfont(item.font) then
5106          quad = font.getfont(item.font).size
5107        end
5108        last_class = class
5109        last_lang = lang
5110      else % if penalty, glue or anything else
5111        last_class = nil
5112      end
5113    end
5114    lang.hyphenate(head)
5115   end
5116 }%
5117 \bbl@luahyphenate}
5118 \gdef\bbl@luahyphenate{%
5119 \let\bbl@luahyphenate\relax
5120 \directlua{
5121   luatexbase.add_to_callback('hyphenate',
5122   function (head, tail)
5123     if Babel.linebreaking.before then
5124       for k, func in ipairs(Babel.linebreaking.before)  do
5125         func(head)
5126       end
5127     end
5128     if Babel.cjk_enabled then
5129       Babel.cjk_linebreak(head)
5130     end
5131     lang.hyphenate(head)
5132     if Babel.linebreaking.after then
5133       for k, func in ipairs(Babel.linebreaking.after)  do
5134         func(head)
5135       end
5136     end
5137     if Babel.sea_enabled then
5138       Babel.sea_disc_to_space(head)
5139     end
5140   end,
5141   'Babel.hyphenate')
5142 }
5143 }
5144 \endgroup
5145 \def\bbl@provide@intraspace{%
5146 \bbl@ifunset{bbl@intsp@\languagename}{}%
5147   {\expandafter\ifx\csname bbl@intsp@\languagename\endcsname\@empty\else
5148     \bbl@xin@{\bbl@cl{lnbrk}}{c}%
5149    \ifin@         % cjk
5150      \bbl@cjkintraspace
5151       \directlua{
```

```
5152              Babel = Babel or {}
5153              Babel.locale_props = Babel.locale_props or {}
5154              Babel.locale_props[\the\localeid].linebreak = 'c'
5155            }%
5156            \bbl@exp{\\\bbl@intraspace\bbl@cl{intsp}\\\@@}%
5157            \ifx\bbl@KVP@intrapenalty\@nil
5158              \bbl@intrapenalty0\@@
5159            \fi
5160          \else            % sea
5161            \bbl@seaintraspace
5162            \bbl@exp{\\\bbl@intraspace\bbl@cl{intsp}\\\@@}%
5163            \directlua{
5164              Babel = Babel or {}
5165              Babel.sea_ranges = Babel.sea_ranges or {}
5166              Babel.set_chranges('\bbl@cl{sbcp}',
5167                                 '\bbl@cl{chrng}')
5168            }%
5169            \ifx\bbl@KVP@intrapenalty\@nil
5170              \bbl@intrapenalty0\@@
5171            \fi
5172          \fi
5173        \fi
5174        \ifx\bbl@KVP@intrapenalty\@nil\else
5175          \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
5176        \fi}}
```

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.
Work in progress.
Common stuff.

```
5177 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
5178 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
5179 \DisableBabelHook{babel-fontspec}
5180 ⟨⟨Font selection⟩⟩
```

13.6   **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.

```
5181 \directlua{
5182 Babel.script_blocks = {
5183   ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
5184              {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
```

```
5185    ['Armn'] = {{0x0530, 0x058F}},
5186    ['Beng'] = {{0x0980, 0x09FF}},
5187    ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
5188    ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
5189    ['Cyrl'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
5190                {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
5191    ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
5192    ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
5193                {0xAB00, 0xAB2F}},
5194    ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
5195    % Don't follow strictly Unicode, which places some Coptic letters in
5196    % the 'Greek and Coptic' block
5197    ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
5198    ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
5199                {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
5200                {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
5201                {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
5202                {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
5203                {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
5204    ['Hebr'] = {{0x0590, 0x05FF}},
5205    ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
5206                {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
5207    ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
5208    ['Knda'] = {{0x0C80, 0x0CFF}},
5209    ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
5210                {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
5211                {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
5212    ['Laoo'] = {{0x0E80, 0x0EFF}},
5213    ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
5214                {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
5215                {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
5216    ['Mahj'] = {{0x11150, 0x1117F}},
5217    ['Mlym'] = {{0x0D00, 0x0D7F}},
5218    ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
5219    ['Orya'] = {{0x0B00, 0x0B7F}},
5220    ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
5221    ['Syrc'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
5222    ['Taml'] = {{0x0B80, 0x0BFF}},
5223    ['Telu'] = {{0x0C00, 0x0C7F}},
5224    ['Tfng'] = {{0x2D30, 0x2D7F}},
5225    ['Thai'] = {{0x0E00, 0x0E7F}},
5226    ['Tibt'] = {{0x0F00, 0x0FFF}},
5227    ['Vaii'] = {{0xA500, 0xA63F}},
5228    ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
5229 }
5230
5231 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
5232 Babel.script_blocks.Hant = Babel.script_blocks.Hans
5233 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
5234
5235 function Babel.locale_map(head)
5236    if not Babel.locale_mapped then return head end
5237
5238    local LOCALE = luatexbase.registernumber'bbl@attr@locale'
5239    local GLYPH = node.id('glyph')
5240    local inmath = false
5241    local toloc_save
5242    for item in node.traverse(head) do
5243        local toloc
```

```
5244    if not inmath and item.id == GLYPH then
5245      % Optimization: build a table with the chars found
5246      if Babel.chr_to_loc[item.char] then
5247        toloc = Babel.chr_to_loc[item.char]
5248      else
5249        for lc, maps in pairs(Babel.loc_to_scr) do
5250          for _, rg in pairs(maps) do
5251            if item.char >= rg[1] and item.char <= rg[2] then
5252              Babel.chr_to_loc[item.char] = lc
5253              toloc = lc
5254              break
5255            end
5256          end
5257        end
5258      end
5259      % Now, take action, but treat composite chars in a different
5260      % fashion, because they 'inherit' the previous locale. Not yet
5261      % optimized.
5262      if not toloc and
5263          (item.char >= 0x0300 and item.char <= 0x036F) or
5264          (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
5265          (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
5266        toloc = toloc_save
5267      end
5268      if toloc and toloc > -1 then
5269        if Babel.locale_props[toloc].lg then
5270          item.lang = Babel.locale_props[toloc].lg
5271          node.set_attribute(item, LOCALE, toloc)
5272        end
5273        if Babel.locale_props[toloc]['/'..item.font] then
5274          item.font = Babel.locale_props[toloc]['/'..item.font]
5275        end
5276        toloc_save = toloc
5277      end
5278    elseif not inmath and item.id == 7 then
5279      item.replace = item.replace and Babel.locale_map(item.replace)
5280      item.pre     = item.pre and Babel.locale_map(item.pre)
5281      item.post    = item.post and Babel.locale_map(item.post)
5282    elseif item.id == node.id'math' then
5283      inmath = (item.subtype == 0)
5284    end
5285  end
5286  return head
5287 end
5288 }
```

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

```
5289 \newcommand\babelcharproperty[1]{%
5290   \count@=#1\relax
5291   \ifvmode
5292     \expandafter\bbl@chprop
5293   \else
5294     \bbl@error{\string\babelcharproperty\space can be used only in\\%
5295                vertical mode (preamble or between paragraphs)}%
5296                {See the manual for futher info}%
5297   \fi}
5298 \newcommand\bbl@chprop[3][\the\count@]{%
5299   \@tempcnta=#1\relax
```

181

```
5300  \bbl@ifunset{bbl@chprop@#2}%
5301    {\bbl@error{No property named '#2'. Allowed values are\\%
5302                direction (bc), mirror (bmg), and linebreak (lb)}%
5303                {See the manual for futher info}}%
5304    {}%
5305  \loop
5306    \bbl@cs{chprop@#2}{#3}%
5307  \ifnum\count@<\@tempcnta
5308    \advance\count@\@ne
5309  \repeat}
5310 \def\bbl@chprop@direction#1{%
5311  \directlua{
5312    Babel.characters[\the\count@] =  Babel.characters[\the\count@] or {}
5313    Babel.characters[\the\count@]['d'] = '#1'
5314  }}
5315 \let\bbl@chprop@bc\bbl@chprop@direction
5316 \def\bbl@chprop@mirror#1{%
5317  \directlua{
5318    Babel.characters[\the\count@] =  Babel.characters[\the\count@] or {}
5319    Babel.characters[\the\count@]['m'] = '\number#1'
5320  }}
5321 \let\bbl@chprop@bmg\bbl@chprop@mirror
5322 \def\bbl@chprop@linebreak#1{%
5323  \directlua{
5324    Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
5325    Babel.cjk_characters[\the\count@]['c'] = '#1'
5326  }}
5327 \let\bbl@chprop@lb\bbl@chprop@linebreak
5328 \def\bbl@chprop@locale#1{%
5329  \directlua{
5330    Babel.chr_to_loc = Babel.chr_to_loc or {}
5331    Babel.chr_to_loc[\the\count@] =
5332      \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@@#1}}\space
5333  }}
```

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.

```
5334 \begingroup
5335 \catcode`\#=12
5336 \catcode`\%=12
5337 \catcode`\&=14
5338 \directlua{
5339  Babel.linebreaking.post_replacements = {}
5340  Babel.linebreaking.pre_replacements = {}
5341
5342  function Babel.str_to_nodes(fn, matches, base)
5343    local n, head, last
```

```
5344      if fn == nil then return nil end
5345      for s in string.utfvalues(fn(matches)) do
5346        if base.id == 7 then
5347          base = base.replace
5348        end
5349        n = node.copy(base)
5350        n.char    = s
5351        if not head then
5352          head = n
5353        else
5354          last.next = n
5355        end
5356        last = n
5357      end
5358      return head
5359    end
5360
5361    function Babel.fetch_word(head, funct)
5362      local word_string = ''
5363      local word_nodes = {}
5364      local lang
5365      local item = head
5366      local inmath = false
5367
5368      while item do
5369
5370        if item.id == 29
5371            and not(item.char == 124) &% ie, not |
5372            and not(item.char == 61)  &% ie, not =
5373            and not inmath
5374            and (item.lang == lang or lang == nil) then
5375          lang = lang or item.lang
5376          word_string = word_string .. unicode.utf8.char(item.char)
5377          word_nodes[#word_nodes+1] = item
5378
5379        elseif item.id == 7 and item.subtype == 2 and not inmath then
5380          word_string = word_string .. '='
5381          word_nodes[#word_nodes+1] = item
5382
5383        elseif item.id == 7 and item.subtype == 3 and not inmath then
5384          word_string = word_string .. '|'
5385          word_nodes[#word_nodes+1] = item
5386
5387        elseif item.id == 11 and item.subtype == 0 then
5388          inmath = true
5389
5390        elseif word_string == '' then
5391          &% pass
5392
5393        else
5394          return word_string, word_nodes, item, lang
5395        end
5396
5397        item = item.next
5398      end
5399    end
5400
5401    function Babel.post_hyphenate_replace(head)
5402      local u = unicode.utf8
```

```
5403     local lbkr = Babel.linebreaking.post_replacements
5404     local word_head = head
5405
5406     while true do
5407       local w, wn, nw, lang = Babel.fetch_word(word_head)
5408       if not lang then return head end
5409
5410       if not lbkr[lang] then
5411         break
5412       end
5413
5414       for k=1, #lbkr[lang] do
5415         local p = lbkr[lang][k].pattern
5416         local r = lbkr[lang][k].replace
5417
5418         while true do
5419           local matches = { u.match(w, p) }
5420           if #matches < 2 then break end
5421
5422           local first = table.remove(matches, 1)
5423           local last =  table.remove(matches, #matches)
5424
5425           &% Fix offsets, from bytes to unicode.
5426           first = u.len(w:sub(1, first-1)) + 1
5427           last  = u.len(w:sub(1, last-1))
5428
5429           local new  &% used when inserting and removing nodes
5430           local changed = 0
5431
5432           &% This loop traverses the replace list and takes the
5433           &% corresponding actions
5434           for q = first, last do
5435             local crep = r[q-first+1]
5436             local char_node = wn[q]
5437             local char_base = char_node
5438
5439             if crep and crep.data then
5440               char_base = wn[crep.data+first-1]
5441             end
5442
5443             if crep == {} then
5444               break
5445             elseif crep == nil then
5446               changed = changed + 1
5447               node.remove(head, char_node)
5448             elseif crep and (crep.pre or crep.no or crep.post) then
5449               changed = changed + 1
5450               d = node.new(7, 0)   &% (disc, discretionary)
5451               d.pre = Babel.str_to_nodes(crep.pre, matches, char_base)
5452               d.post = Babel.str_to_nodes(crep.post, matches, char_base)
5453               d.replace = Babel.str_to_nodes(crep.no, matches, char_base)
5454               d.attr = char_base.attr
5455               if crep.pre == nil then  &% TeXbook p96
5456                 d.penalty  = crep.penalty or tex.hyphenpenalty
5457               else
5458                 d.penalty  = crep.penalty or tex.exhyphenpenalty
5459               end
5460               head, new = node.insert_before(head, char_node, d)
5461               node.remove(head, char_node)
```

```
5462                if q == 1 then
5463                  word_head = new
5464                end
5465              elseif crep and crep.string then
5466                changed = changed + 1
5467                local str = crep.string(matches)
5468                if str == '' then
5469                  if q == 1 then
5470                    word_head = char_node.next
5471                  end
5472                  head, new = node.remove(head, char_node)
5473                elseif char_node.id == 29 and u.len(str) == 1 then
5474                  char_node.char = string.utfvalue(str)
5475                else
5476                  local n
5477                  for s in string.utfvalues(str) do
5478                    if char_node.id == 7 then
5479                      log('Automatic hyphens cannot be replaced, just removed.')
5480                    else
5481                      n = node.copy(char_base)
5482                    end
5483                    n.char = s
5484                    if q == 1 then
5485                      head, new = node.insert_before(head, char_node, n)
5486                      word_head = new
5487                    else
5488                      node.insert_before(head, char_node, n)
5489                    end
5490                  end
5491
5492                  node.remove(head, char_node)
5493                end  &% string length
5494              end  &% if char and char.string
5495            end  &% for char in match
5496            if changed > 20 then
5497              texio.write('Too many changes. Ignoring the rest.')
5498            elseif changed > 0 then
5499              w, wn, nw = Babel.fetch_word(word_head)
5500            end
5501
5502        end  &% for match
5503      end  &% for patterns
5504      word_head = nw
5505    end  &% for words
5506    return head
5507  end
5508
5509  &%%%
5510  &% Preliminary code for \babelprehyphenation
5511  &% TODO. Copypaste pattern. Merge with fetch_word
5512  function Babel.fetch_subtext(head, funct)
5513    local word_string = ''
5514    local word_nodes = {}
5515    local lang
5516    local item = head
5517    local inmath = false
5518
5519    while item do
5520
```

```
5521      if item.id == 29 then
5522        local locale = node.get_attribute(item, Babel.attr_locale)
5523
5524        if not(item.char == 124) &% ie, not | = space
5525            and not inmath
5526            and (locale == lang or lang == nil) then
5527          lang = lang or locale
5528          word_string = word_string .. unicode.utf8.char(item.char)
5529          word_nodes[#word_nodes+1] = item
5530        end
5531
5532        if item == node.tail(head) then
5533          item = nil
5534          return word_string, word_nodes, item, lang
5535        end
5536
5537      elseif item.id == 12 and item.subtype == 13 and not inmath then
5538        word_string = word_string .. '|'
5539        word_nodes[#word_nodes+1] = item
5540
5541        if item == node.tail(head) then
5542          item = nil
5543          return word_string, word_nodes, item, lang
5544        end
5545
5546      elseif item.id == 11 and item.subtype == 0 then
5547          inmath = true
5548
5549      elseif word_string == '' then
5550        &% pass
5551
5552      else
5553        return word_string, word_nodes, item, lang
5554      end
5555
5556      item = item.next
5557    end
5558  end
5559
5560  &% TODO. Copypaste pattern. Merge with pre_hyphenate_replace
5561  function Babel.pre_hyphenate_replace(head)
5562    local u = unicode.utf8
5563    local lbkr = Babel.linebreaking.pre_replacements
5564    local word_head = head
5565
5566    while true do
5567      local w, wn, nw, lang = Babel.fetch_subtext(word_head)
5568      if not lang then return head end
5569
5570      if not lbkr[lang] then
5571        break
5572      end
5573
5574      for k=1, #lbkr[lang] do
5575        local p = lbkr[lang][k].pattern
5576        local r = lbkr[lang][k].replace
5577
5578        while true do
5579          local matches = { u.match(w, p) }
```

```
5580          if #matches < 2 then break end
5581
5582          local first = table.remove(matches, 1)
5583          local last =  table.remove(matches, #matches)
5584
5585          &% Fix offsets, from bytes to unicode.
5586          first = u.len(w:sub(1, first-1)) + 1
5587          last  = u.len(w:sub(1, last-1))
5588
5589          local new  &% used when inserting and removing nodes
5590          local changed = 0
5591
5592          &% This loop traverses the replace list and takes the
5593          &% corresponding actions
5594          for q = first, last do
5595            local crep = r[q-first+1]
5596            local char_node = wn[q]
5597            local char_base = char_node
5598
5599            if crep and crep.data then
5600              char_base = wn[crep.data+first-1]
5601            end
5602
5603            if crep == {} then
5604              break
5605            elseif crep == nil then
5606              changed = changed + 1
5607              node.remove(head, char_node)
5608            elseif crep and crep.string then
5609              changed = changed + 1
5610              local str = crep.string(matches)
5611              if str == '' then
5612                if q == 1 then
5613                  word_head = char_node.next
5614                end
5615                head, new = node.remove(head, char_node)
5616              elseif char_node.id == 29 and u.len(str) == 1 then
5617                char_node.char = string.utfvalue(str)
5618              else
5619                local n
5620                for s in string.utfvalues(str) do
5621                  if char_node.id == 7 then
5622                    log('Automatic hyphens cannot be replaced, just removed.')
5623                  else
5624                    n = node.copy(char_base)
5625                  end
5626                  n.char = s
5627                  if q == 1 then
5628                    head, new = node.insert_before(head, char_node, n)
5629                    word_head = new
5630                  else
5631                    node.insert_before(head, char_node, n)
5632                  end
5633                end
5634
5635                node.remove(head, char_node)
5636              end  &% string length
5637            end  &% if char and char.string
5638          end  &% for char in match
```

187

```
5639          if changed > 20 then
5640            texio.write('Too many changes. Ignoring the rest.')
5641          elseif changed > 0 then
5642            &% For one-to-one can we modifiy directly the
5643            &% values without re-fetching? Very likely.
5644            w, wn, nw = Babel.fetch_subtext(word_head)
5645          end
5646
5647        end  &% for match
5648      end  &% for patterns
5649      word_head = nw
5650    end  &% for words
5651    return head
5652  end
5653  &%%% end of preliminary code for \babelprehyphenation
5654
5655  &% The following functions belong to the next macro
5656
5657  &% This table stores capture maps, numbered consecutively
5658  Babel.capture_maps = {}
5659
5660  function Babel.capture_func(key, cap)
5661    local ret = "[[" .. cap:gsub('{([0-9])}', "]]..m[%1]..[[") .. "]]"
5662    ret = ret:gsub('{([0-9])|([^|]+)|(.-)}', Babel.capture_func_map)
5663    ret = ret:gsub("%[%[%]%]%.%.", '')
5664    ret = ret:gsub("%.%.%[%[%]%]", '')
5665    return key .. [[=function(m) return ]] .. ret .. [[ end]]
5666  end
5667
5668  function Babel.capt_map(from, mapno)
5669    return Babel.capture_maps[mapno][from] or from
5670  end
5671
5672  &% Handle the {n|abc|ABC} syntax in captures
5673  function Babel.capture_func_map(capno, from, to)
5674    local froms = {}
5675    for s in string.utfcharacters(from) do
5676      table.insert(froms, s)
5677    end
5678    local cnt = 1
5679    table.insert(Babel.capture_maps, {})
5680    local mlen = table.getn(Babel.capture_maps)
5681    for s in string.utfcharacters(to) do
5682      Babel.capture_maps[mlen][froms[cnt]] = s
5683      cnt = cnt + 1
5684    end
5685    return "]]..Babel.capt_map(m[" .. capno .. "]," ..
5686           (mlen) .. ").." .. "[["
5687  end
5688 }
```

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).

```
5689 \catcode`\#=6
5690 \gdef\babelposthyphenation#1#2#3{&%
5691   \bbl@activateposthyphen
5692   \begingroup
5693     \def\babeltempa{\bbl@add@list\babeltempb}&%
5694     \let\babeltempb\@empty
5695     \bbl@foreach{#3}{&%
5696       \bbl@ifsamestring{##1}{remove}&%
5697         {\bbl@add@list\babeltempb{nil}}&%
5698         {\directlua{
5699            local rep = [[##1]]
5700            rep = rep:gsub(     '(no)%s*=%s*([^%s,]*)', Babel.capture_func)
5701            rep = rep:gsub(    '(pre)%s*=%s*([^%s,]*)', Babel.capture_func)
5702            rep = rep:gsub(   '(post)%s*=%s*([^%s,]*)', Babel.capture_func)
5703            rep = rep:gsub('(string)%s*=%s*([^%s,]*)', Babel.capture_func)
5704            tex.print([[\string\babeltempa{{]] .. rep .. [[}}]])
5705          }}}&%
5706     \directlua{
5707       local lbkr = Babel.linebreaking.post_replacements
5708       local u = unicode.utf8
5709       &% Convert pattern:
5710       local patt = string.gsub([==[#2]==], '%s', '')
5711       if not u.find(patt, '()', nil, true) then
5712         patt = '()' .. patt .. '()'
5713       end
5714       patt = string.gsub(patt, '%(%)%^', '^()')
5715       patt = string.gsub(patt, '%$%(%)', '()$')
5716       texio.write('***************' .. patt)
5717       patt = u.gsub(patt, '{(.)}',
5718               function (n)
5719                 return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
5720               end)
5721       lbkr[\the\csname l@#1\endcsname] = lbkr[\the\csname l@#1\endcsname] or {}
5722       table.insert(lbkr[\the\csname l@#1\endcsname],
5723                 { pattern = patt, replace = { \babeltempb } })
5724     }&%
5725   \endgroup}
5726 % TODO. Working !!! Copypaste pattern.
5727 \gdef\babelprehyphenation#1#2#3{&%
5728   \bbl@activateprehyphen
5729   \begingroup
5730     \def\babeltempa{\bbl@add@list\babeltempb}&%
5731     \let\babeltempb\@empty
5732     \bbl@foreach{#3}{&%
5733       \bbl@ifsamestring{##1}{remove}&%
5734         {\bbl@add@list\babeltempb{nil}}&%
5735         {\directlua{
5736            local rep = [[##1]]
5737            rep = rep:gsub('(string)%s*=%s*([^%s,]*)', Babel.capture_func)
5738            tex.print([[\string\babeltempa{{]] .. rep .. [[}}]])
5739          }}}&%
5740     \directlua{
5741       local lbkr = Babel.linebreaking.pre_replacements
5742       local u = unicode.utf8
5743       &% Convert pattern:
5744       local patt = string.gsub([==[#2]==], '%s', '')
```

189

```
5745        if not u.find(patt, '()', nil, true) then
5746          patt = '()' .. patt .. '()'
5747        end
5748        patt = u.gsub(patt, '{(.)}',
5749                    function (n)
5750                      return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
5751                    end)
5752        lbkr[\the\csname bbl@id@@#1\endcsname] = lbkr[\the\csname  bbl@id@@#1\endcsname] or {}
5753        table.insert(lbkr[\the\csname bbl@id@@#1\endcsname],
5754                    { pattern = patt, replace = { \babeltempb } })
5755      }&%
5756   \endgroup}
5757 \endgroup
5758 \def\bbl@activateposthyphen{%
5759   \let\bbl@activateposthyphen\relax
5760   \directlua{
5761     Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
5762   }}
5763 % TODO. Working !!!
5764 \def\bbl@activateprehyphen{%
5765   \let\bbl@activateprehyphen\relax
5766   \directlua{
5767     Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
5768   }}
```

13.7 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.

```
5769 \bbl@trace{Redefinitions for bidi layout}
5770 \ifx\@eqnnum\@undefined\else
5771   \ifx\bbl@attr@dir\@undefined\else
5772     \edef\@eqnnum{{%
5773       \unexpanded{\ifcase\bbl@attr@dir\else\bbl@textdir\@ne\fi}%
5774       \unexpanded\expandafter{\@eqnnum}}}
5775   \fi
5776 \fi
5777 \ifx\bbl@opt@layout\@nnil\endinput\fi  % if no layout
5778 \ifnum\bbl@bidimode>\z@
5779   \def\bbl@nextfake#1{%  non-local changes, use always inside a group!
5780     \bbl@exp{%
5781       \mathdir\the\bodydir
5782       #1%                 Once entered in math, set boxes to restore values
5783       \<ifmmode>%
5784         \everyvbox{%
5785           \the\everyvbox
5786           \bodydir\the\bodydir
```

```
5787          \mathdir\the\mathdir
5788          \everyhbox{\the\everyhbox}%
5789          \everyvbox{\the\everyvbox}}%
5790        \everyhbox{%
5791          \the\everyhbox
5792          \bodydir\the\bodydir
5793          \mathdir\the\mathdir
5794          \everyhbox{\the\everyhbox}%
5795          \everyvbox{\the\everyvbox}}%
5796      \<fi>}}%
5797    \def\@hangfrom#1{%
5798      \setbox\@tempboxa\hbox{{#1}}%
5799      \hangindent\wd\@tempboxa
5800      \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
5801        \shapemode\@ne
5802      \fi
5803      \noindent\box\@tempboxa}
5804 \fi
5805 \IfBabelLayout{tabular}
5806    {\let\bbl@OL@@tabular\@tabular
5807     \bbl@replace\@tabular{$}{\bbl@nextfake$}%
5808     \let\bbl@NL@@tabular\@tabular
5809     \AtBeginDocument{%
5810       \ifx\bbl@NL@@tabular\@tabular\else
5811         \bbl@replace\@tabular{$}{\bbl@nextfake$}%
5812         \let\bbl@NL@@tabular\@tabular
5813       \fi}}
5814    {}
5815 \IfBabelLayout{lists}
5816    {\let\bbl@OL@list\list
5817     \bbl@sreplace\list{\parshape}{\bbl@listparshape}%
5818     \let\bbl@NL@list\list
5819     \def\bbl@listparshape#1#2#3{%
5820       \parshape #1 #2 #3 %
5821       \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
5822         \shapemode\tw@
5823       \fi}}
5824    {}
5825 \IfBabelLayout{graphics}
5826    {\let\bbl@pictresetdir\relax
5827     \def\bbl@pictsetdir{%
5828       \ifcase\bbl@thetextdir
5829         \let\bbl@pictresetdir\relax
5830       \else
5831         \textdir TLT\relax
5832         \def\bbl@pictresetdir{\textdir TRT\relax}%
5833       \fi}%
5834     \let\bbl@OL@@picture\@picture
5835     \let\bbl@OL@put\put
5836     \bbl@sreplace\@picture{\hskip-}{\bbl@pictsetdir\hskip-}%
5837     \def\put(#1,#2)#3{%  Not easy to patch. Better redefine.
5838       \@killglue
5839       \raise#2\unitlength
5840       \hb@xt@\z@{\kern#1\unitlength{\bbl@pictresetdir#3}\hss}}%
5841     \AtBeginDocument
5842       {\ifx\tikz@atbegin@node\@undefined\else
5843         \let\bbl@OL@pgfpicture\pgfpicture
5844         \bbl@sreplace\pgfpicture{\pgfpicturetrue}%
5845           {\bbl@pictsetdir\pgfpicturetrue}%
```

```
5846              \bbl@add\pgfsys@beginpicture{\bbl@pictsetdir}%
5847              \bbl@add\tikz@atbegin@node{\bbl@pictresetdir}%
5848          \fi}}
5849    {}
```

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`.

```
5850 \IfBabelLayout{counters}%
5851    {\let\bbl@OL@@textsuperscript\@textsuperscript
5852     \bbl@sreplace\@textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
5853     \let\bbl@latinarabic=\@arabic
5854     \let\bbl@OL@@arabic\@arabic
5855     \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
5856     \@ifpackagewith{babel}{bidi=default}%
5857       {\let\bbl@asciiroman=\@roman
5858        \let\bbl@OL@@roman\@roman
5859        \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciiroman#1}}}%
5860        \let\bbl@asciiRoman=\@Roman
5861        \let\bbl@OL@@roman\@Roman
5862        \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}%
5863        \let\bbl@OL@labelenumii\labelenumii
5864        \def\labelenumii{)\theenumii(}%
5865        \let\bbl@OL@p@enumiii\p@enumiii
5866        \def\p@enumiii{\p@enumii)\theenumii(}}{}}{}
5867 ⟨⟨Footnote changes⟩⟩
5868 \IfBabelLayout{footnotes}%
5869    {\let\bbl@OL@footnote\footnote
5870     \BabelFootnote\footnote\languagename{}{}%
5871     \BabelFootnote\localfootnote\languagename{}{}%
5872     \BabelFootnote\mainfootnote{}{}{}}
5873    {}
```

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.

```
5874 \IfBabelLayout{extras}%
5875    {\let\bbl@OL@underline\underline
5876     \bbl@sreplace\underline{$\@@underline}{\bbl@nextfake$\@@underline}%
5877     \let\bbl@OL@LaTeX2e\LaTeX2e
5878     \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
5879       \if b\expandafter\@car\f@series\@nil\boldmath\fi
5880       \babelsublr{%
5881         \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}}}
5882    {}
5883 ⟨/luatex⟩
```

13.8   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.

```
5884 ⟨∗basic-r⟩
5885 Babel = Babel or {}
5886
5887 Babel.bidi_enabled = true
5888
5889 require('babel-data-bidi.lua')
5890
5891 local characters = Babel.characters
5892 local ranges = Babel.ranges
5893
5894 local DIR = node.id("dir")
5895
5896 local function dir_mark(head, from, to, outer)
5897   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
5898   local d = node.new(DIR)
5899   d.dir = '+' .. dir
5900   node.insert_before(head, from, d)
5901   d = node.new(DIR)
5902   d.dir = '-' .. dir
5903   node.insert_after(head, to, d)
5904 end
5905
5906 function Babel.bidi(head, ispar)
5907   local first_n, last_n           -- first and last char with nums
```

```
5908   local last_es                    -- an auxiliary 'last' used with nums
5909   local first_d, last_d            -- first and last char in L/R block
5910   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):

```
5911   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
5912   local strong_lr = (strong == 'l') and 'l' or 'r'
5913   local outer = strong
5914
5915   local new_dir = false
5916   local first_dir = false
5917   local inmath = false
5918
5919   local last_lr
5920
5921   local type_n = ''
5922
5923   for item in node.traverse(head) do
5924
5925     -- three cases: glyph, dir, otherwise
5926     if item.id == node.id'glyph'
5927       or (item.id == 7 and item.subtype == 2) then
5928
5929       local itemchar
5930       if item.id == 7 and item.subtype == 2 then
5931         itemchar = item.replace.char
5932       else
5933         itemchar = item.char
5934       end
5935       local chardata = characters[itemchar]
5936       dir = chardata and chardata.d or nil
5937       if not dir then
5938         for nn, et in ipairs(ranges) do
5939           if itemchar < et[1] then
5940             break
5941           elseif itemchar <= et[2] then
5942             dir = et[3]
5943             break
5944           end
5945         end
5946       end
5947       dir = dir or 'l'
5948       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).

```
5949       if new_dir then
5950         attr_dir = 0
5951         for at in node.traverse(item.attr) do
5952           if at.number == luatexbase.registernumber'bbl@attr@dir' then
5953             attr_dir = at.value % 3
5954           end
5955         end
```

194

```
5956        if attr_dir == 1 then
5957          strong = 'r'
5958        elseif attr_dir == 2 then
5959          strong = 'al'
5960        else
5961          strong = 'l'
5962        end
5963        strong_lr = (strong == 'l') and 'l' or 'r'
5964        outer = strong_lr
5965        new_dir = false
5966      end
5967
5968      if dir == 'nsm' then dir = strong end              -- W1
```

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

```
5969      dir_real = dir              -- We need dir_real to set strong below
5970      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:

```
5971      if strong == 'al' then
5972        if dir == 'en' then dir = 'an' end                -- W2
5973        if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
5974        strong_lr = 'r'                                   -- W3
5975      end
```

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

```
5976    elseif item.id == node.id'dir' and not inmath then
5977      new_dir = true
5978      dir = nil
5979    elseif item.id == node.id'math' then
5980      inmath = (item.subtype == 0)
5981    else
5982      dir = nil          -- Not a char
5983    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>.

```
5984    if dir == 'en' or dir == 'an' or dir == 'et' then
5985      if dir ~= 'et' then
5986        type_n = dir
5987      end
5988      first_n = first_n or item
5989      last_n = last_es or item
5990      last_es = nil
5991    elseif dir == 'es' and last_n then -- W3+W6
5992      last_es = item
5993    elseif dir == 'cs' then              -- it's right - do nothing
5994    elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
5995      if strong_lr == 'r' and type_n ~= '' then
5996        dir_mark(head, first_n, last_n, 'r')
5997      elseif strong_lr == 'l' and first_d and type_n == 'an' then
5998        dir_mark(head, first_n, last_n, 'r')
```

```
5999          dir_mark(head, first_d, last_d, outer)
6000          first_d, last_d = nil, nil
6001        elseif strong_lr == 'l' and type_n ~= '' then
6002          last_d = last_n
6003        end
6004        type_n = ''
6005        first_n, last_n = nil, nil
6006      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:

```
6007      if dir == 'l' or dir == 'r' then
6008        if dir ~= outer then
6009          first_d = first_d or item
6010          last_d = item
6011        elseif first_d and dir ~= strong_lr then
6012          dir_mark(head, first_d, last_d, outer)
6013          first_d, last_d = nil, nil
6014        end
6015      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.

```
6016      if dir and not last_lr and dir ~= 'l' and outer == 'r' then
6017        item.char = characters[item.char] and
6018                    characters[item.char].m or item.char
6019      elseif (dir or new_dir) and last_lr ~= item then
6020        local mir = outer .. strong_lr .. (dir or outer)
6021        if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
6022          for ch in node.traverse(node.next(last_lr)) do
6023            if ch == item then break end
6024            if ch.id == node.id'glyph' and characters[ch.char] then
6025              ch.char = characters[ch.char].m or ch.char
6026            end
6027          end
6028        end
6029      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).

```
6030      if dir == 'l' or dir == 'r' then
6031        last_lr = item
6032        strong = dir_real           -- Don't search back - best save now
6033        strong_lr = (strong == 'l') and 'l' or 'r'
6034      elseif new_dir then
6035        last_lr = nil
6036      end
6037    end
```

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

```
6038  if last_lr and outer == 'r' then
6039    for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
6040      if characters[ch.char] then
```

```
6041         ch.char = characters[ch.char].m or ch.char
6042       end
6043     end
6044   end
6045   if first_n then
6046     dir_mark(head, first_n, last_n, outer)
6047   end
6048   if first_d then
6049     dir_mark(head, first_d, last_d, outer)
6050   end
```

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

```
6051   return node.prev(head) or head
6052 end
6053 ⟨/basic-r⟩
```

And here the Lua code for bidi=basic:

```
6054 ⟨∗basic⟩
6055 Babel = Babel or {}
6056
6057 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
6058
6059 Babel.fontmap = Babel.fontmap or {}
6060 Babel.fontmap[0] = {}      -- l
6061 Babel.fontmap[1] = {}      -- r
6062 Babel.fontmap[2] = {}      -- al/an
6063
6064 Babel.bidi_enabled = true
6065 Babel.mirroring_enabled = true
6066
6067 require('babel-data-bidi.lua')
6068
6069 local characters = Babel.characters
6070 local ranges = Babel.ranges
6071
6072 local DIR = node.id('dir')
6073 local GLYPH = node.id('glyph')
6074
6075 local function insert_implicit(head, state, outer)
6076   local new_state = state
6077   if state.sim and state.eim and state.sim ~= state.eim then
6078     dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
6079     local d = node.new(DIR)
6080     d.dir = '+' .. dir
6081     node.insert_before(head, state.sim, d)
6082     local d = node.new(DIR)
6083     d.dir = '-' .. dir
6084     node.insert_after(head, state.eim, d)
6085   end
6086   new_state.sim, new_state.eim = nil, nil
6087   return head, new_state
6088 end
6089
6090 local function insert_numeric(head, state)
6091   local new
6092   local new_state = state
6093   if state.san and state.ean and state.san ~= state.ean then
6094     local d = node.new(DIR)
```

197

```
6095     d.dir = '+TLT'
6096     _, new = node.insert_before(head, state.san, d)
6097     if state.san == state.sim then state.sim = new end
6098     local d = node.new(DIR)
6099     d.dir = '-TLT'
6100     _, new = node.insert_after(head, state.ean, d)
6101     if state.ean == state.eim then state.eim = new end
6102   end
6103   new_state.san, new_state.ean = nil, nil
6104   return head, new_state
6105 end
6106
6107 -- TODO - \hbox with an explicit dir can lead to wrong results
6108 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
6109 -- was s made to improve the situation, but the problem is the 3-dir
6110 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
6111 -- well.
6112
6113 function Babel.bidi(head, ispar, hdir)
6114   local d    -- d is used mainly for computations in a loop
6115   local prev_d = ''
6116   local new_d = false
6117
6118   local nodes = {}
6119   local outer_first = nil
6120   local inmath = false
6121
6122   local glue_d = nil
6123   local glue_i = nil
6124
6125   local has_en = false
6126   local first_et = nil
6127
6128   local ATDIR = luatexbase.registernumber'bbl@attr@dir'
6129
6130   local save_outer
6131   local temp = node.get_attribute(head, ATDIR)
6132   if temp then
6133     temp = temp % 3
6134     save_outer = (temp == 0 and 'l') or
6135                  (temp == 1 and 'r') or
6136                  (temp == 2 and 'al')
6137   elseif ispar then          -- Or error? Shouldn't happen
6138     save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
6139   else                       -- Or error? Shouldn't happen
6140     save_outer = ('TRT' == hdir) and 'r' or 'l'
6141   end
6142     -- when the callback is called, we are just _after_ the box,
6143     -- and the textdir is that of the surrounding text
6144   -- if not ispar and hdir ~= tex.textdir then
6145   --   save_outer = ('TRT' == hdir) and 'r' or 'l'
6146   -- end
6147   local outer = save_outer
6148   local last = outer
6149   -- 'al' is only taken into account in the first, current loop
6150   if save_outer == 'al' then save_outer = 'r' end
6151
6152   local fontmap = Babel.fontmap
6153
```

```
6154   for item in node.traverse(head) do
6155
6156     -- In what follows, #node is the last (previous) node, because the
6157     -- current one is not added until we start processing the neutrals.
6158
6159     -- three cases: glyph, dir, otherwise
6160     if item.id == GLYPH
6161        or (item.id == 7 and item.subtype == 2) then
6162
6163       local d_font = nil
6164       local item_r
6165       if item.id == 7 and item.subtype == 2 then
6166         item_r = item.replace    -- automatic discs have just 1 glyph
6167       else
6168         item_r = item
6169       end
6170       local chardata = characters[item_r.char]
6171       d = chardata and chardata.d or nil
6172       if not d or d == 'nsm' then
6173         for nn, et in ipairs(ranges) do
6174           if item_r.char < et[1] then
6175             break
6176           elseif item_r.char <= et[2] then
6177             if not d then d = et[3]
6178             elseif d == 'nsm' then d_font = et[3]
6179             end
6180             break
6181           end
6182         end
6183       end
6184       d = d or 'l'
6185
6186       -- A short 'pause' in bidi for mapfont
6187       d_font = d_font or d
6188       d_font = (d_font == 'l' and 0) or
6189                (d_font == 'nsm' and 0) or
6190                (d_font == 'r' and 1) or
6191                (d_font == 'al' and 2) or
6192                (d_font == 'an' and 2) or nil
6193       if d_font and fontmap and fontmap[d_font][item_r.font] then
6194         item_r.font = fontmap[d_font][item_r.font]
6195       end
6196
6197       if new_d then
6198         table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
6199         if inmath then
6200           attr_d = 0
6201         else
6202           attr_d = node.get_attribute(item, ATDIR)
6203           attr_d = attr_d % 3
6204         end
6205         if attr_d == 1 then
6206           outer_first = 'r'
6207           last = 'r'
6208         elseif attr_d == 2 then
6209           outer_first = 'r'
6210           last = 'al'
6211         else
6212           outer_first = 'l'
```

```
6213          last = 'l'
6214        end
6215        outer = last
6216        has_en = false
6217        first_et = nil
6218        new_d = false
6219      end
6220
6221    if glue_d then
6222      if (d == 'l' and 'l' or 'r') ~= glue_d then
6223          table.insert(nodes, {glue_i, 'on', nil})
6224      end
6225      glue_d = nil
6226      glue_i = nil
6227    end
6228
6229  elseif item.id == DIR then
6230    d = nil
6231    new_d = true
6232
6233  elseif item.id == node.id'glue' and item.subtype == 13 then
6234    glue_d = d
6235    glue_i = item
6236    d = nil
6237
6238  elseif item.id == node.id'math' then
6239    inmath = (item.subtype == 0)
6240
6241  else
6242    d = nil
6243  end
6244
6245  -- AL <= EN/ET/ES      -- W2 + W3 + W6
6246  if last == 'al' and d == 'en' then
6247    d = 'an'            -- W3
6248  elseif last == 'al' and (d == 'et' or d == 'es') then
6249    d = 'on'           -- W6
6250  end
6251
6252  -- EN + CS/ES + EN       -- W4
6253  if d == 'en' and #nodes >= 2 then
6254    if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
6255        and nodes[#nodes-1][2] == 'en' then
6256      nodes[#nodes][2] = 'en'
6257    end
6258  end
6259
6260  -- AN + CS + AN         -- W4 too, because uax9 mixes both cases
6261  if d == 'an' and #nodes >= 2 then
6262    if (nodes[#nodes][2] == 'cs')
6263        and nodes[#nodes-1][2] == 'an' then
6264      nodes[#nodes][2] = 'an'
6265    end
6266  end
6267
6268  -- ET/EN                 -- W5 + W7->l / W6->on
6269  if d == 'et' then
6270    first_et = first_et or (#nodes + 1)
6271  elseif d == 'en' then
```

```
6272        has_en = true
6273        first_et = first_et or (#nodes + 1)
6274      elseif first_et then        -- d may be nil here !
6275        if has_en then
6276          if last == 'l' then
6277            temp = 'l'      -- W7
6278          else
6279            temp = 'en'     -- W5
6280          end
6281        else
6282          temp = 'on'       -- W6
6283        end
6284        for e = first_et, #nodes do
6285          if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
6286        end
6287        first_et = nil
6288        has_en = false
6289      end
6290
6291      if d then
6292        if d == 'al' then
6293          d = 'r'
6294          last = 'al'
6295        elseif d == 'l' or d == 'r' then
6296          last = d
6297        end
6298        prev_d = d
6299        table.insert(nodes, {item, d, outer_first})
6300      end
6301
6302      outer_first = nil
6303
6304    end
6305
6306    -- TODO -- repeated here in case EN/ET is the last node. Find a
6307    -- better way of doing things:
6308    if first_et then        -- dir may be nil here !
6309      if has_en then
6310        if last == 'l' then
6311          temp = 'l'      -- W7
6312        else
6313          temp = 'en'     -- W5
6314        end
6315      else
6316        temp = 'on'       -- W6
6317      end
6318      for e = first_et, #nodes do
6319        if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
6320      end
6321    end
6322
6323    -- dummy node, to close things
6324    table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
6325
6326    -------------- NEUTRAL -----------------
6327
6328    outer = save_outer
6329    last = outer
6330
```

```
6331   local first_on = nil
6332
6333   for q = 1, #nodes do
6334     local item
6335
6336     local outer_first = nodes[q][3]
6337     outer = outer_first or outer
6338     last = outer_first or last
6339
6340     local d = nodes[q][2]
6341     if d == 'an' or d == 'en' then d = 'r' end
6342     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
6343
6344     if d == 'on' then
6345       first_on = first_on or q
6346     elseif first_on then
6347       if last == d then
6348         temp = d
6349       else
6350         temp = outer
6351       end
6352       for r = first_on, q - 1 do
6353         nodes[r][2] = temp
6354         item = nodes[r][1]     -- MIRRORING
6355         if Babel.mirroring_enabled and item.id == GLYPH
6356             and temp == 'r' and characters[item.char] then
6357           local font_mode = font.fonts[item.font].properties.mode
6358           if font_mode ~= 'harf' and font_mode ~= 'plug' then
6359             item.char = characters[item.char].m or item.char
6360           end
6361         end
6362       end
6363       first_on = nil
6364     end
6365
6366     if d == 'r' or d == 'l' then last = d end
6367   end
6368
6369   --------------   IMPLICIT, REORDER ----------------
6370
6371   outer = save_outer
6372   last = outer
6373
6374   local state = {}
6375   state.has_r = false
6376
6377   for q = 1, #nodes do
6378
6379     local item = nodes[q][1]
6380
6381     outer = nodes[q][3] or outer
6382
6383     local d = nodes[q][2]
6384
6385     if d == 'nsm' then d = last end               -- W1
6386     if d == 'en' then d = 'an' end
6387     local isdir = (d == 'r' or d == 'l')
6388
6389     if outer == 'l' and d == 'an' then
```

```
6390        state.san = state.san or item
6391        state.ean = item
6392      elseif state.san then
6393        head, state = insert_numeric(head, state)
6394      end
6395
6396      if outer == 'l' then
6397        if d == 'an' or d == 'r' then      -- im -> implicit
6398          if d == 'r' then state.has_r = true end
6399          state.sim = state.sim or item
6400          state.eim = item
6401        elseif d == 'l' and state.sim and state.has_r then
6402          head, state = insert_implicit(head, state, outer)
6403        elseif d == 'l' then
6404          state.sim, state.eim, state.has_r = nil, nil, false
6405        end
6406      else
6407        if d == 'an' or d == 'l' then
6408          if nodes[q][3] then -- nil except after an explicit dir
6409            state.sim = item  -- so we move sim 'inside' the group
6410          else
6411            state.sim = state.sim or item
6412          end
6413          state.eim = item
6414        elseif d == 'r' and state.sim then
6415          head, state = insert_implicit(head, state, outer)
6416        elseif d == 'r' then
6417          state.sim, state.eim = nil, nil
6418        end
6419      end
6420
6421      if isdir then
6422        last = d             -- Don't search back - best save now
6423      elseif d == 'on' and state.san  then
6424        state.san = state.san or item
6425        state.ean = item
6426      end
6427
6428    end
6429
6430    return node.prev(head) or head
6431  end
6432 ⟨/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.

```
6433 ⟨*nil⟩
6434 \ProvidesLanguage{nil}[⟨⟨date⟩⟩ ⟨⟨version⟩⟩ Nil language]
6435 \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.

```
6436 \ifx\l@nil\@undefined
6437   \newlanguage\l@nil
6438   \@namedef{bbl@hyphendata@\the\l@nil}{{}{}}% Remove warning
6439   \let\bbl@elt\relax
6440   \edef\bbl@languages{%  Add it to the list of languages
6441     \bbl@languages\bbl@elt{nil}{\the\l@nil}{}{}}
6442 \fi
```

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

```
6443 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}
```

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

\captionnil
\datenil
```
6444 \let\captionsnil\@empty
6445 \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.

```
6446 \ldf@finish{nil}
6447 ⟨/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.

6448 ⟨∗bplain | blplain⟩
6449 \catcode`\{=1 % left brace is begin-group character
6450 \catcode`\}=2 % right brace is end-group character
6451 \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).

6452 \openin 0 hyphen.cfg
6453 \ifeof0
6454 \else
6455   \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.

6456   \def\input #1 {%
6457   \let\input\a
6458   \a hyphen.cfg
6459   \let\a\undefined
6460   }
6461 \fi
6462 ⟨/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.

6463 ⟨bplain⟩\a plain.tex
6464 ⟨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.

6465 ⟨bplain⟩\def\fmtname{babel-plain}
6466 ⟨blplain⟩\def\fmtname{babel-lplain}

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

16.2 Emulating some LaTeX features

The following code duplicates or emulates parts of LaTeX 2$_\varepsilon$ that are needed for babel.

6467 ⟨⟨∗Emulate LaTeX⟩⟩ ≡
6468   % == Code for plain ==
6469 \def\@empty{}
6470 \def\loadlocalcfg#1{%
6471   \openin0#1.cfg
6472   \ifeof0
6473   \closein0
6474   \else
6475   \closein0
6476   {\immediate\write16{***********************************}%
6477   \immediate\write16{* Local config file #1.cfg used}%
6478   \immediate\write16{*}%
6479   }
6480   \input #1.cfg\relax
6481   \fi
6482   \@endofldf}

205

16.3 General tools

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

```
6483 \long\def\@firstofone#1{#1}
6484 \long\def\@firstoftwo#1#2{#1}
6485 \long\def\@secondoftwo#1#2{#2}
6486 \def\@nnil{\@nil}
6487 \def\@gobbletwo#1#2{}
6488 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
6489 \def\@star@or@long#1{%
6490   \@ifstar
6491   {\let\l@ngrel@x\relax#1}%
6492   {\let\l@ngrel@x\long#1}}
6493 \let\l@ngrel@x\relax
6494 \def\@car#1#2\@nil{#1}
6495 \def\@cdr#1#2\@nil{#2}
6496 \let\@typeset@protect\relax
6497 \let\protected@edef\edef
6498 \long\def\@gobble#1{}
6499 \edef\@backslashchar{\expandafter\@gobble\string\\}
6500 \def\strip@prefix#1>{}
6501 \def\g@addto@macro#1#2{{%
6502     \toks@\expandafter{#1#2}%
6503     \xdef#1{\the\toks@}}}
6504 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
6505 \def\@nameuse#1{\csname #1\endcsname}
6506 \def\@ifundefined#1{%
6507   \expandafter\ifx\csname#1\endcsname\relax
6508     \expandafter\@firstoftwo
6509   \else
6510     \expandafter\@secondoftwo
6511   \fi}
6512 \def\@expandtwoargs#1#2#3{%
6513   \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
6514 \def\zap@space#1 #2{%
6515   #1%
6516   \ifx#2\@empty\else\expandafter\zap@space\fi
6517   #2}
6518 \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}`.

```
6519 \ifx\@preamblecmds\@undefined
6520   \def\@preamblecmds{}
6521 \fi
6522 \def\@onlypreamble#1{%
6523   \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
6524     \@preamblecmds\do#1}}
6525 \@onlypreamble\@onlypreamble
```

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

```
6526 \def\begindocument{%
6527   \@begindocumenthook
6528   \global\let\@begindocumenthook\@undefined
6529   \def\do##1{\global\let##1\@undefined}%
6530   \@preamblecmds
6531   \global\let\do\noexpand}
```

```
6532 \ifx\@begindocumenthook\@undefined
6533  \def\@begindocumenthook{}
6534 \fi
6535 \@onlypreamble\@begindocumenthook
6536 \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.

```
6537 \def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
6538 \@onlypreamble\AtEndOfPackage
6539 \def\@endofldf{}
6540 \@onlypreamble\@endofldf
6541 \let\bbl@afterlang\@empty
6542 \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.

```
6543 \catcode`\&=\z@
6544 \ifx&if@filesw\@undefined
6545  \expandafter\let\csname if@filesw\expandafter\endcsname
6546    \csname iffalse\endcsname
6547 \fi
6548 \catcode`\&=4
```

Mimick LaTeX's commands to define control sequences.

```
6549 \def\newcommand{\@star@or@long\new@command}
6550 \def\new@command#1{%
6551  \@testopt{\@newcommand#1}0}
6552 \def\@newcommand#1[#2]{%
6553  \@ifnextchar [{\@xargdef#1[#2]}%
6554                {\@argdef#1[#2]}}
6555 \long\def\@argdef#1[#2]#3{%
6556  \@yargdef#1\@ne{#2}{#3}}
6557 \long\def\@xargdef#1[#2][#3]#4{%
6558  \expandafter\def\expandafter#1\expandafter{%
6559    \expandafter\@protected@testopt\expandafter #1%
6560    \csname\string#1\expandafter\endcsname{#3}}%
6561  \expandafter\@yargdef \csname\string#1\endcsname
6562  \tw@{#2}{#4}}
6563 \long\def\@yargdef#1#2#3{%
6564  \@tempcnta#3\relax
6565  \advance \@tempcnta \@ne
6566  \let\@hash@\relax
6567  \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
6568  \@tempcntb #2%
6569  \@whilenum\@tempcntb <\@tempcnta
6570  \do{%
6571    \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
6572    \advance\@tempcntb \@ne}%
6573  \let\@hash@##%
6574  \l@ngrel@x\expandafter\def\expandafter#1\reserved@a}
6575 \def\providecommand{\@star@or@long\provide@command}
6576 \def\provide@command#1{%
6577  \begingroup
6578    \escapechar\m@ne\xdef\@gtempa{{\string#1}}%
6579  \endgroup
6580  \expandafter\@ifundefined\@gtempa
6581    {\def\reserved@a{\new@command#1}}%
```

```
6582     {\let\reserved@a\relax
6583      \def\reserved@a{\new@command\reserved@a}}%
6584    \reserved@a}%
6585 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
6586 \def\declare@robustcommand#1{%
6587    \edef\reserved@a{\string#1}%
6588    \def\reserved@b{#1}%
6589    \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
6590    \edef#1{%
6591       \ifx\reserved@a\reserved@b
6592          \noexpand\x@protect
6593          \noexpand#1%
6594       \fi
6595       \noexpand\protect
6596       \expandafter\noexpand\csname
6597          \expandafter\@gobble\string#1 \endcsname
6598    }%
6599    \expandafter\new@command\csname
6600       \expandafter\@gobble\string#1 \endcsname
6601 }
6602 \def\x@protect#1{%
6603    \ifx\protect\@typeset@protect\else
6604       \@x@protect#1%
6605    \fi
6606 }
6607 \catcode`\&=\z@  % Trick to hide conditionals
6608    \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.

```
6609    \def\bbl@tempa{\csname newif\endcsname&ifin@}
6610 \catcode`\&=4
6611 \ifx\in@\@undefined
6612    \def\in@#1#2{%
6613      \def\in@@##1#1##2##3\in@@{%
6614         \ifx\in@##2\in@false\else\in@true\fi}%
6615      \in@@#2#1\in@\in@@}
6616 \else
6617    \let\bbl@tempa\@empty
6618 \fi
6619 \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).

```
6620 \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.

```
6621 \def\@ifl@aded#1#2#3#4{}
```

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

```
6622 \ifx\@tempcnta\@undefined
6623   \csname newcount\endcsname\@tempcnta\relax
6624 \fi
6625 \ifx\@tempcntb\@undefined
6626   \csname newcount\endcsname\@tempcntb\relax
6627 \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).

```
6628 \ifx\bye\@undefined
6629   \advance\count10 by -2\relax
6630 \fi
6631 \ifx\@ifnextchar\@undefined
6632   \def\@ifnextchar#1#2#3{%
6633     \let\reserved@d=#1%
6634     \def\reserved@a{#2}\def\reserved@b{#3}%
6635     \futurelet\@let@token\@ifnch}
6636   \def\@ifnch{%
6637     \ifx\@let@token\@sptoken
6638       \let\reserved@c\@xifnch
6639     \else
6640       \ifx\@let@token\reserved@d
6641         \let\reserved@c\reserved@a
6642       \else
6643         \let\reserved@c\reserved@b
6644       \fi
6645     \fi
6646     \reserved@c}
6647   \def\:{\let\@sptoken= } \:  % this makes \@sptoken a space token
6648   \def\:{\@xifnch} \expandafter\def\: {\futurelet\@let@token\@ifnch}
6649 \fi
6650 \def\@testopt#1#2{%
6651   \@ifnextchar[{#1}{#1[#2]}}
6652 \def\@protected@testopt#1{%
6653   \ifx\protect\@typeset@protect
6654     \expandafter\@testopt
6655   \else
6656     \@x@protect#1%
6657   \fi}
6658 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
6659     #2\relax}\fi}
6660 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
6661        \else\expandafter\@gobble\fi{#1}}
```

16.4 Encoding related macros

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

```
6662 \def\DeclareTextCommand{%
6663   \@dec@text@cmd\providecommand
6664 }
6665 \def\ProvideTextCommand{%
6666   \@dec@text@cmd\providecommand
6667 }
6668 \def\DeclareTextSymbol#1#2#3{%
6669   \@dec@text@cmd\chardef#1{#2}#3\relax
6670 }
6671 \def\@dec@text@cmd#1#2#3{%
6672   \expandafter\def\expandafter#2%
```

```
6673          \expandafter{%
6674             \csname#3-cmd\expandafter\endcsname
6675             \expandafter#2%
6676             \csname#3\string#2\endcsname
6677          }%
6678 %    \let\@ifdefinable\@rc@ifdefinable
6679     \expandafter#1\csname#3\string#2\endcsname
6680 }
6681 \def\@current@cmd#1{%
6682    \ifx\protect\@typeset@protect\else
6683        \noexpand#1\expandafter\@gobble
6684    \fi
6685 }
6686 \def\@changed@cmd#1#2{%
6687    \ifx\protect\@typeset@protect
6688        \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
6689           \expandafter\ifx\csname ?\string#1\endcsname\relax
6690              \expandafter\def\csname ?\string#1\endcsname{%
6691                 \@changed@x@err{#1}%
6692              }%
6693           \fi
6694           \global\expandafter\let
6695             \csname\cf@encoding \string#1\expandafter\endcsname
6696             \csname ?\string#1\endcsname
6697        \fi
6698        \csname\cf@encoding\string#1%
6699           \expandafter\endcsname
6700    \else
6701        \noexpand#1%
6702    \fi
6703 }
6704 \def\@changed@x@err#1{%
6705     \errhelp{Your command will be ignored, type <return> to proceed}%
6706     \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
6707 \def\DeclareTextCommandDefault#1{%
6708    \DeclareTextCommand#1?%
6709 }
6710 \def\ProvideTextCommandDefault#1{%
6711    \ProvideTextCommand#1?%
6712 }
6713 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
6714 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
6715 \def\DeclareTextAccent#1#2#3{%
6716    \DeclareTextCommand#1{#2}[1]{\accent#3 ##1}
6717 }
6718 \def\DeclareTextCompositeCommand#1#2#3#4{%
6719    \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
6720    \edef\reserved@b{\string##1}%
6721    \edef\reserved@c{%
6722      \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
6723    \ifx\reserved@b\reserved@c
6724        \expandafter\expandafter\expandafter\ifx
6725           \expandafter\@car\reserved@a\relax\relax\@nil
6726           \@text@composite
6727        \else
6728           \edef\reserved@b##1{%
6729              \def\expandafter\noexpand
6730                 \csname#2\string#1\endcsname####1{%
6731                 \noexpand\@text@composite
```

210

```
6732                    \expandafter\noexpand\csname#2\string#1\endcsname
6733                    ####1\noexpand\@empty\noexpand\@text@composite
6734                    {##1}%
6735              }%
6736          }%
6737          \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
6738      \fi
6739      \expandafter\def\csname\expandafter\string\csname
6740          #2\endcsname\string#1-\string#3\endcsname{#4}
6741    \else
6742      \errhelp{Your command will be ignored, type <return> to proceed}%
6743      \errmessage{\string\DeclareTextCompositeCommand\space used on
6744          inappropriate command \protect#1}
6745    \fi
6746 }
6747 \def\@text@composite#1#2#3\@text@composite{%
6748    \expandafter\@text@composite@x
6749        \csname\string#1-\string#2\endcsname
6750 }
6751 \def\@text@composite@x#1#2{%
6752    \ifx#1\relax
6753        #2%
6754    \else
6755        #1%
6756    \fi
6757 }
6758 %
6759 \def\@strip@args#1:#2-#3\@strip@args{#2}
6760 \def\DeclareTextComposite#1#2#3#4{%
6761    \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
6762    \bgroup
6763        \lccode`\@=#4%
6764        \lowercase{%
6765    \egroup
6766        \reserved@a @%
6767    }%
6768 }
6769 %
6770 \def\UseTextSymbol#1#2{#2}
6771 \def\UseTextAccent#1#2#3{}
6772 \def\@use@text@encoding#1{}
6773 \def\DeclareTextSymbolDefault#1#2{%
6774    \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
6775 }
6776 \def\DeclareTextAccentDefault#1#2{%
6777    \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
6778 }
6779 \def\cf@encoding{OT1}
```

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

```
6780 \DeclareTextAccent{\"}{OT1}{127}
6781 \DeclareTextAccent{\'}{OT1}{19}
6782 \DeclareTextAccent{\^}{OT1}{94}
6783 \DeclareTextAccent{\`}{OT1}{18}
6784 \DeclareTextAccent{\~}{OT1}{126}
```

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

```
6785 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
```

```
6786 \DeclareTextSymbol{\textquotedblright}{OT1}{`\"}
6787 \DeclareTextSymbol{\textquoteleft}{OT1}{`\`}
6788 \DeclareTextSymbol{\textquoteright}{OT1}{`\'}
6789 \DeclareTextSymbol{\i}{OT1}{16}
6790 \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`.

```
6791 \ifx\scriptsize\@undefined
6792   \let\scriptsize\sevenrm
6793 \fi
6794   % End of code for plain
6795 ⟨⟨/Emulate LaTeX⟩⟩
```

A proxy file:

```
6796 ⟨*plain⟩
6797 \input babel.def
6798 ⟨/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).