# Babel

The standard distribution of LaTeX contains a number of document classes that are meant to be used, but also serve as examples for other users to create their own document classes. These document classes have become very popular among LaTeX users. But it should be kept in mind that they were designed for American tastes and typography. At one time they even contained a number of hard-wired texts.

This manual describes babel, a package that makes use of the capabilities of TeX version 3 and, to some extent, xetex and luatex, to provide an environment in which documents can be typeset in a language other than US English, or in more than one language or script.

Current development is focused on Unicode engines (XeTeX and LuaTeX) and the so-called *complex scripts*. New features related to font selection, bidi writing and the like will be added incrementally.

Babel provides support (total or partial) for about 200 languages, either as a "classical" package option or as an `ini` file. Furthermore, new languages can be created from scratch easily.

# Contents

# Troubleshoooting

# Part I
# User guide

- This user guide focuses on LaTeX. There are also some notes on its use with Plain TeX.

- Changes and new features with relation to version 3.8 are highlighted with New X.XX . The most recent features could be still unstable. Please, report any issues you find.

- If you are interested in the TeX multilingual support, please join the kadingira list on `http://tug.org/mailman/listinfo/kadingira`. You can follow the development of babel on `https://github.com/latex3/latex2e/tree/master/required/babel` (which provides some sample files, too).

- See section 3.1 for contributing a language.

## 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 standand mechanism for this purpose, namely, passing that language as an optional argument. In addition, you may want to set the font and input encodings.

**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:

```
\documentclass{article}

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

\usepackage[french]{babel}

\begin{document}

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

\end{document}
```

**TROUBLESHOOTING**  A common source of trouble is a wrong setting of the input encoding. Very often you will get the following somewhat cryptic error:

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

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

Another approach is making the language (french in the example) a global option in order to let other packages detect and use it:

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

In this last example, the package varioref will also see the option and will be able to use it.

**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.

## 1.2  Multilingual documents

In multilingual documents, just use several options. The last one is considered the main language, 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:

```
\documentclass{article}
\usepackage[main=english,dutch]{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, decribed 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.

```
\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}
```

## 1.3  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 accept 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 is a more general mechanism.

## 1.4  xelatex and lualatex

Many languages are compatible with xetex and luatex. With them you can use babel to localize the documents.
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**  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.

---

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

```
\documentclass{article}

\usepackage[vietnamese,danish]{babel}

\begin{document}

\prefacename{} -- \alsoname{} -- \today

\selectlanguage{vietnamese}

\prefacename{} -- \alsoname{} -- \today

\end{document}
```

**EXAMPLE**  Here is a simple monolingual document in Russian (text from the Wikipedia).
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{article}

\usepackage[russian]{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}
```

## 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 `LANG'. Either you have misspelled
(babel)                its name, it has not been installed, or you requested
(babel)                it in a previous run. Fix its name, install it or just
(babel)                rerun the file, respectively
```

---

[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".

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:

```
\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.

**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 | {⟨*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).

## 1.8   Auxiliary language selectors

| | |
|---|---|
| \begin{otherlanguage} | {⟨*language*⟩}   …   \end{otherlanguage} |

The environment otherlanguage does basically the same as \selectlanguage, except the 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*} | {⟨*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 ypesetting 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.

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

The environment hyphenrules can be used to select *only* he hyphenation rules to be used (it can be used as command, oo). 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 he 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*⟩`}` o 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:

---

[4]Even in the babel kernel there were some macros not compatible with plain. Hopefully these issues will be fixed soon.

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

```
\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, as 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 an 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:

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`).

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. If a character is not known to be a shorthand character its category code will be left unchanged.

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.

\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 could start with, say:

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

However, 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 could then set:

```
\defineshorthand[*polish,*portugese]{"-}{\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 overriden 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.

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

The command \aliasshorthand can be used to let another character perform the same functions as the default shorthand character. If one prefers for example to use the character / over " in typing Polish texts, this can be achieved by entering \aliasshorthand{"}{/}.

**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.

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

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

**\babelshorthand**  {⟨*shorthand*⟩}

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

---

[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

**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]

## 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 availabe 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

---

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

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 – of course, in such a case you cannot use shorthands in these macros, but this is not a real problem (just use "allowed" characters).

**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 any more.

**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 | main | select | other | other*`

---

[9]You can use alternatively the package silence.

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]

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=

New 3.14 Selects the bidi algorithm to be used in luatex and xetex. See sec. 1.20.

layout=

New 3.16 Selects which layout elements are adapted in bidi documents. See sec. 1.20.

### 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 hyphenations 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 hyphenations 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
```

---

[10]Turned off in plain.

[11]Duplicated options count as several ones.

[12]Providing foreign is pointless, because the case mapping applied is that at the end of 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.

```
    \let\macro\relax}
  \usepackage[foo,bar]{babel}
```

## 1.13   ini **files**

An alternative approach to define a language is by means of an ini file. Currently babel
provides about 200 of these files containing the basic data required for a language.
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
currently (by means of \babelprovide), but a higher interface, based on package options,
in under development.

**EXAMPLE**  Although Georgian has its own ldf file, here is how to declare this language
      with an ini file in Unicode engines. The nil language is required, because currently
      babel raises an error if there is no language.

```
\documentclass{book}

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

\babelfont{rm}{DejaVu Sans}

\begin{document}

\tableofcontents

\chapter{სამზარეულო და სუფრის ტრადიციები}

ქართული ტრადიციული სამზარეულო ერთ-ერთი უმდიდრესია მთელ მსოფლიოში.

\end{document}
```

Here is the list (u means Unicode captions, and l means LICR captions):

| | | | |
|---|---|---|---|
| af | Afrikaans[ul] | bm | Bambara |
| agq | Aghem | bn | Bangla[ul] |
| ak | Akan | bo | Tibetan[u] |
| am | Amharic[ul] | brx | Bodo |
| ar | Arabic[ul] | bs-Cyrl | Bosnian |
| as | Assamese | bs-Latn | Bosnian[ul] |
| asa | Asu | bs | Bosnian[ul] |
| ast | Asturian[ul] | ca | Catalan[ul] |
| az-Cyrl | Azerbaijani | ce | Chechen |
| az-Latn | Azerbaijani | cgg | Chiga |
| az | Azerbaijani[ul] | chr | Cherokee |
| bas | Basaa | ckb | Central Kurdish |
| be | Belarusian[ul] | cs | Czech[ul] |
| bem | Bemba | cy | Welsh[ul] |
| bez | Bena | da | Danish[ul] |
| bg | Bulgarian[ul] | dav | Taita |

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

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

---

aghem
akan
albanian
american
amharic
arabic
armenian
assamese
asturian
asu
australian
austrian
azerbaijani-cyrillic
azerbaijani-cyrl
azerbaijani-latin
azerbaijani-latn
azerbaijani
bafia
bambara
basaa
basque
belarusian
bemba
bena
bengali
bodo
bosnian-cyrillic
bosnian-cyrl
bosnian-latin
bosnian-latn
bosnian
brazilian
breton
british
bulgarian
burmese
canadian
cantonese
catalan
centralatlastamazight
centralkurdish
chechen
cherokee
chiga
chinese-hans-hk
chinese-hans-mo
chinese-hans-sg
chinese-hans
chinese-hant-hk
chinese-hant-mo

chinese-hant
chinese-simplified-hongkongsarchina
chinese-simplified-macausarchina
chinese-simplified-singapore
chinese-simplified
chinese-traditional-hongkongsarchina
chinese-traditional-macausarchina
chinese-traditional
chinese
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
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
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
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

### 1.14   Selecting fonts

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

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

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. Alternatively, you may set a font for a script – just precede its name (lowercase) with a star (eg, *devanagari). 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.

```
\documentclass{article}

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

\babelprovide[import=he]{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 could replace the red line above with, say:

```
\babelfont{rm}{Iwona}
\babelfont[hebrew]{rm}{FreeSerif}
```

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

**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:

---

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

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

This makes sure the OpenType script for Devanagari is deva and not dev2.

**NOTE**  Directionality is a property affecting margins, intentation, column order, etc., not just text. Therefore, it is under the direct control of the language, which appplies both the script and the direction to the text. As a consequence, there is no need to set Script when declaring a font (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 are 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 could 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**  Do not use \set*xxxx*font and \babelfont at the same time. \babelfont follows the standard LaTeX conventions to set the basic families – define \\*xx*default, and activate it with \\*xx*family. On the other hand, \set*xxxx*font in fontspec takes a different approach, because \\*xx*family is redefined with the family name hardcoded (so that \\*xx*default becomes no-op). Of course, both methods are incompatible, and if you use \set*xxxx*font, font switching with \babelfont just does *not* work (nor the standard \\*xx*default, for that matter).

## 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 it.

- 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*⟩.

**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.

## 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.

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

Defines the internal structure of the language with some defaults: the hyphen rules, if not
available, are set to the current ones, left and right hyphen mins are set to 2 and 3, but
captions and date are not defined. Conveniently, babel warns you about what to do. Very
likely you will find alerts like that in the log file:

```
Package babel Warning: \mylangchaptername not set. Please, define
(babel)                it 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.

**EXAMPLE**  If you need a language named arhinish:

```
    \usepackage[danish]{babel}
    \babelprovide{arhinish}
    \renewcommand\arhinishchaptername{Chapitula}
    \renewcommand\arhinishrefname{Refirenke}
    \renewcommand\arhinishhyphenmins{22}
```

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, date, and hyphenmins. 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.
There are about 200 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

25

files. A few languages will show a warning about the current lack of suitability of the date format (hindi, french, breton, and occitan).

Besides \today, there is a \<language>date macro with three arguments: year, month and day numbers. In fact, \today calls \<language>today, which in turn calls \<language>date{\the\year}{\the\month}{\the\day}.

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 supresses hyphenation (because the pattern list is empty).

main   This valueless option makes the language the main one. Only in newly defined languages.

script=   ⟨*script-name*⟩

New 3.15   Sets the script name to be used by fontspec (eg, Devanagari). Overrides the value in the ini file. This value is particularly important because it sets the writing direction.

language=   ⟨*language-name*⟩

New 3.15   Sets the language name to be used by fontspec (eg, Hindi). Overrides the value in the ini file. Not so important, but sometimes still relevant.

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

## 1.17   Getting the current language name

`\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 TEX sense, 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.

>   **WARNING**  The advice about `\languagename` also applies here – use iflang instead of
>   `\iflanguage` if possible.

## 1.18   Hyphenation tools

`\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 oportunity or, in TEX terms, a "discretionary"; a *hard hyphen* is a hyphen with a breaking oportunity after it. A further type is a *non-breaking hyphen*, a hyphen without a breaking oportunity.
In TEX, `-` and `\-` forbid further breaking oportunities 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, Portugese, 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 oportunities in the rest of the word. Therefore, some macros are provide 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, Portugese and Spanish.

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

- `\babelhyphen{empty}` inserts a break oportunity 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 enabling 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.

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

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

## 1.19   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.[15]
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.[16]

\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

---

[14]With luatex exceptions and patterns can be modified almost freely. However, this is very likely a task for a separate package and babel only provides the most basic tools.
[15]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.
[16]But still defined for backwards compatibility.

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.

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

## 1.20   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 could 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**   Setting bidi text has many subtleties (see for example <https://www.w3.org/TR/html-bidi/>). *This means the babel bidi code may take some time before it is truly stable.*[17] 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).

There are some package options controlling bidi writing.

bidi=   default | basic-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 by 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.

> **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, which will be improved in the future. Remember basic-r is available in luatex only.[18]

```
\documentclass{article}

\usepackage[nil, bidi=basic-r]{babel}

\babelprovide[import=ar, main]{arabic}

\babelfont{rm}{FreeSerif}

\begin{document}
```

---

[17] A basic stable version for luatex is planned before Summer 2018. Other engines must wait very likely until Winter.

[18] At the time of this writing some Arabic fonts are not rendered correctly by the default luatex font loader, with misplaced kerns inside some words, so double check the resulting text. It seems a fix is on the way, but in the meanwhile you could have a look at the workaround available on GitHub, under /required/babel/samples

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

New 3.16   *To be expanded.* Selects which layout elements are adapted in bidi documents.
You may use several options with a comma-separated list (eg,
layout=counters.contents.sectioning). This list will be expanded in future releases
(tables, captions, etc.). 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 to reorder correctly 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;
lists  required in xetex and pdftex, but only in multilingual documents in luatex;
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 multilingual
   documents in all engines; you may use alternatively \BabelFootnote described below
   (what this options 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). New 3.18


\babelsublr   {⟨*lr-text*⟩}

Digits in pdftex must be marked up explicitly (unlike luatex with bidi=basic-r and,
usually, xetex). Mainly for it (although available in all engines, because it can be useful),
this command is provided to set {⟨*lr-text*⟩} in L mode. 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.


\BabelPatchSection   {⟨*section-name*⟩}

Mainly for bidi text, but it could 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, but with this command you can set them individually if necessary (but note then tocs and marks are not touched).

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

New 3.17  Something like:

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

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

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

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

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

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

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

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

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

## 1.21  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 settting options are also used (eg, `\ProsodicMarksOn` in latin).

## 1.22  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  {⟨*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).

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 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 loads switch.def. It can be used to load a different version of this files or to load nothing.

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).

## 1.23 Languages supported by babel

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**  bahasa, indonesian, indon, bahasai
**Interlingua**  interlingua
**Irish Gaelic**  irish
**Italian**  italian
**Latin**  latin
**Lower Sorbian**  lowersorbian
**Malay**  bahasam, malay, melayu
**North Sami**  samin
**Norwegian**  norsk, nynorsk
**Polish**  polish
**Portuguese**  portuges, portuguese, brazilian, brazil
**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). 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.

### 1.24   Tips, workarounds, know 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 could revert to using uppercase labels, you can use `\lowercase{\ref{foo}}` inside the argument of `\chapter`, or, if you will not use shorthands in labels, set the `safe` option to `none` or `bib`.

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

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

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

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

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

  (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.[19] So, if you write a chunk of French text with `\foreinglanguage`, the apostrophes might not be taken into account. This is a limitation of TeX, not of babel. Alternatively, you may use `\useshorthands` to activate `'` and `\defineshorthand`, or redefine `\textquoteright` (the latter is called by the non-ASCII right quote).

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

- 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).

---

[19]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.

- 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.

## 1.25   Current and future work

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).
It is possible now to typeset Arabic or Hebrew with numbers and L text. Next on the roadmap are line breaking in Thai and the like, as well as "non-European" digits. Also on the roadmap are R layouts (lists, footnotes, tables, column order), page and section numbering, and maybe kashida justification.
As to Thai line breaking, here is the basic idea of what luatex can do for us, with the Thai patterns and a little script (the final version will not be so little, of course). It replaces each discretionary by the equivalent to ZWJ.

```
\documentclass{article}

\usepackage[nil]{babel}

\babelprovide[import=th, main]{thai}

\babelfont{rm}{FreeSerif}

\directlua{
local GLYF = node.id'glyph'
function insertsp (head)
  local size = 0
  for item in node.traverse(head) do
    local i = item.id
    if i == GLYF then
      f = font.getfont(item.font)
      size = f.size
    elseif i == 7 then
      local n = node.new(12, 0)
      node.setglue(n, 0, size * 1) % 1 is a factor
      node.insert_before(head, item, n)
```

```
      node.remove(head, item)
    end
  end
end

luatexbase.add_to_callback('hyphenate',
  function (head, tail)
    lang.hyphenate(head)
    insertsp(head)
  end, 'insertsp')
}

\begin{document}

(Thai text.)

\end{document}
```

Useful additions would be, for example, time, currency, addresses and personal names.[20].
But that is the easy part, because they don't require modifying the LaTeX internals.

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.

## 1.26 Tentative and experimental code

**Option** `bidi=basic`

New 3.19   With this package option *both* L and R text can be mixed without explicit markup (which 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 accomplised with an option in `\babelprovide`, as illustrated:

```
\documentclass{book}

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

\babelprovide[mapfont=direction]{arabic}

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

\begin{document}

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

\end{document}
```

---

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

What `mapfont=direction` means is, 'when a character has the same direction as the script for the "provided" language (`arabic` in this case), then change its font to that set for this language (here defined indirectly via `*arabic`)'.

There is a sample on GitHub, under `/required/babel/samples`.

**Old stuff**

Handling of **"Unicode" fonts** is problematic. There is fontspec, but special macros are required (not only the NFSS ones) and it doesn't provide "orthogonal axis" for features, including those related to the language (mainly language and script). A couple of tentative macros, were provided by babel ($\geq$3.9g) with a partial solution. These macros are now deprecated – use `\babelfont`.

- `\babelFSstore{`⟨*babel-language*⟩`}` sets the current three basic families (rm, sf, tt) as the default for the language given.

- `\babelFSdefault{`⟨*babel-language*⟩`}{`⟨*fontspec-features*⟩`}` patches `\fontspec` so that the given features are always passed as the optional argument or added to it (not an ideal solution).

So, for example:

```
\setmainfont[Language=Turkish]{Minion Pro}
\babelFSstore{turkish}
\setmainfont{Minion Pro}
\babelFSfeatures{turkish}{Language=Turkish}
```

**Bidi writing** is taking its *first steps. First steps* means exactly that. On the other hand, xetex poses quite different challenges. Document layout (lists, footnotes, etc.) is not touched at all by default, but an option to manage them is almost finished (for luatex). See the code section for `\foreignlanguage*` (a new starred version of `\foreignlanguage`). xetex relies on the font to properly handle these unmarked changes, so it is not under the control of TeX.

## 2   Loading languages with `language.dat`

TeX and most engines based on it (pdfTeX, xetex, $\epsilon$-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).[21] 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).[22]

---

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

[22]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[23]. 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.[24] For example:

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

With the previous settings, if the enconding 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 could 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.

---

[23]This is because different operating systems sometimes use *very* different file-naming conventions.
[24]This in 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 unknow 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.[25]

- 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.

---

[25]But not removed, for backward compatibility.

## 3.1 Guidelines for contributed languages

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: `http://www.texnia.com/incubator.html`. If your 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. For older versions of `plain.tex` and `lplain.tex` a substitute definition is used. Here "language" is used in the TeX sense of set of hyphenation patterns.

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

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

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

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

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

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

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

\extras⟨lang⟩  The macro `\extras⟨lang⟩` contains all the extra definitions needed for a specific language.

This macro, like the following, is a hook – you can add things to it, but it must not be used directly.

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

\bbl@declare@ttribute    This is a command to be used in the language definition files for declaring a language attribute. It takes three arguments: the name of the language, the attribute to be defined, and the code to be executed when the attribute is to be used.

\main@language    To postpone the activation of the definitions needed for a language until the beginning of a document, all language definition files should use \main@language instead of \selectlanguage. This will just store the name of the language, and the proper language will be activated at the start of the document.

\ProvidesLanguage    The macro \ProvidesLanguage should be used to identify the language definition files. Its syntax is similar to the syntax of the LaTeX command \ProvidesPackage.

\LdfInit    The macro \LdfInit performs a couple of standard checks that must be made at the beginning of a language definition file, such as checking the category code of the @-sign, preventing the .ldf file from being processed twice, etc.

\ldf@quit    The macro \ldf@quit does work needed if a .ldf file was processed earlier. This includes resetting the category code of the @-sign, preparing the language to be activated at \begin{document} time, and ending the input stream.

\ldf@finish    The macro \ldf@finish does work needed at the end of each .ldf file. This includes resetting the category code of the @-sign, loading a local configuration file, and preparing the language to be activated at \begin{document} time.

\loadlocalcfg    After processing a language definition file, LaTeX can be instructed to load a local configuration file. This file can, for instance, be used to add strings to \captions⟨*lang*⟩ to support local document classes. The user will be informed that this configuration file has been loaded. This macro is called by \ldf@finish.

\substitutefontfamily    (Deprecated.) This command takes three arguments, a font encoding and two font family names. It creates a font description file for the first font in the given encoding. This .fd file will instruct LaTeX to use a font from the second family when a font from the first family in the given encoding seems to be needed.

### 3.3  Skeleton

Here is the basic structure of an ldf file, with a language, a dialect and an attribute. Strings are best defined using the method explained in 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>}
```

## 3.4   Support for active characters

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

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

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

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

\bbl@add@special   The TeXbook states: "Plain TeX includes a macro called \dospecials that is essentially a set
\bbl@remove@special   macro, representing the set of all characters that have a special category code." [2, 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[26].

\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 could be either global (usually) or local (sometimes). That does not seem very consistent, but this behavior is preserved for backward compatibility. If you are using etoolbox, by Philipp Lehman, consider using the tools provided by this package instead of \addto.

## 3.7  Macros common to a number of languages

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

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

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

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

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

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

## 3.8  Encoding-dependent strings

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

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

---

[26]This mechanism was introduced by Bernd Raichle.

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.

<div>

**\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 take precedence (ie, it works much like \providecommand).

Encoding info is charset= followed by a charset, which if given sets how the strings should be traslated to the internal representation used by the engine, typically utf8, which is the only value supported currently (default is no traslations). 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 honoured (in a encoded way).

The ⟨*category*⟩ is either captions, date or extras. You must stick to these three categories, even if no error is raised when using other name.[27] It may be empty, too, but in such a case using \SetString is an error (but not \SetCase).

</div>

```
\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:

---

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

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

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

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

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

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

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

\EndBabelCommands
```

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

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

The starred version just forces strings to take a value – if not set as package option, then the default for the engine is used. This is not done by default to prevent backward incompatibilities, but if you are creating a new language this version is better. It's up to the maintainers of the current languages to decide if using it is appropiate.[28]

\EndBabelCommands    Marks the end of the series of blocks.

\AfterBabelCommands    {⟨*code*⟩}

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

---

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

| | |
|---|---|
| `\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 be typically 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 intented for minor readjustments only. For example, as T1 is the default case mapping in LaTeX, we could set for Turkish:

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

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

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

\EndBabelCommands
```

(Note the mapping for OT1 is not complete.)

| | |
|---|---|
| `\SetHyphenMap` | `{⟨to-lower-macros⟩}` |

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

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

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

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

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

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

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

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

# 4 Changes

## 4.1 Changes in babel version 3.9

Most of changes in version 3.9 are 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 could 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.

## 4.2  Changes in babel version 3.7

In babel version 3.7 a number of bugs that were found in version 3.6 are fixed. Also a number of changes and additions have occurred:

- Shorthands are expandable again. The disadvantage is that one has to type `'{}a` when the acute accent is used as a shorthand character. The advantage is that a number of other problems (such as the breaking of ligatures, etc.) have vanished.

- Two new commands, `\shorthandon` and `\shorthandoff` have been introduced to enable to temporarily switch off one or more shorthands.

- Support for typesetting Hebrew (and potential support for typesetting other right-to-left written languages) is now available thanks to Rama Porrat and Boris Lavva.

- A language attribute has been added to the `\mark...` commands in order to make sure that a Greek header line comes out right on the last page before a language switch.

- Hyphenation pattern files are now read *inside a group*; therefore any changes a pattern file needs to make to lowercase codes, uppercase codes, and category codes are kept local to that group. If they are needed for the language, these changes will need to be repeated and stored in `\extras...`

- The concept of language attributes is introduced. It is intended to give the user some control over the features a language-definition file provides. Its first use is for the Greek language, where the user can choose the πολυτονικό ("polytonikó" or multi-accented) Greek way of typesetting texts.

- The environment hyphenrules is introduced.

- The syntax of the file `language.dat` has been extended to allow (optionally) specifying the font encoding to be used while processing the patterns file.

- The command `\providehyphenmins` should now be used in language definition files in order to be able to keep any settings provided by the pattern file.

**Part II**
# The 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 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. By default it also loads `switch.def`.

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 $\langle\langle name\rangle\rangle$. That brings a little bit of literate programming.

```
1 ⟨⟨version=3.18.1236⟩⟩
2 ⟨⟨date=2018/04/04⟩⟩
```

# 6  Tools

**Do not use the following macros in ldf files. They may change in the future**. This applies mainly to those recently added for replacing, trimming and looping. The older ones, like `\bbl@afterfi`, will not change.

We define some basic macros which just make the code cleaner. `\bbl@add` is now used internally instead of `\addto` because of the unpredictable behavior of the latter. Used in `babel.def` and in `babel.sty`, which means in LaTeX is executed twice, but we need them when defining options and `babel.def` cannot be load until options have been defined. This does not hurt, but should be fixed somehow.

```
 3 ⟨⟨*Basic macros⟩⟩ ≡
 4 \bbl@trace{Basic macros}
 5 \def\bbl@stripslash{\expandafter\@gobble\string}
 6 \def\bbl@add#1#2{%
 7   \bbl@ifunset{\bbl@stripslash#1}%
 8     {\def#1{#2}}%
 9     {\expandafter\def\expandafter#1\expandafter{#1#2}}}
10 \def\bbl@xin@{\@expandtwoargs\in@}
11 \def\bbl@csarg#1#2{\expandafter#1\csname bbl@#2\endcsname}%
12 \def\bbl@cs#1{\csname bbl@#1\endcsname}
13 \def\bbl@loop#1#2#3{\bbl@@loop#1{#3}#2,\@nnil,}
14 \def\bbl@loopx#1#2{\expandafter\bbl@loop\expandafter#1\expandafter{#2}}
15 \def\bbl@@loop#1#2#3,{%
16   \ifx\@nnil#3\relax\else
17     \def#1{#3}#2\bbl@afterfi\bbl@@loop#1{#2}%
18   \fi}
19 \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.

```
20 \def\bbl@add@list#1#2{%
21   \edef#1{%
22     \bbl@ifunset{\bbl@stripslash#1}%
23       {}%
24       {\ifx#1\@empty\else#1,\fi}%
25     #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[29]. These macros will break if another `\if...\fi` statement appears in one of the arguments and it is not enclosed in braces.

---

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

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

\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.

```
28 \def\bbl@tempa#1{%
29   \long\def\bbl@trim##1##2{%
30     \futurelet\bbl@trim@a\bbl@trim@c##2\@nil\@nil##1\@nil\relax{##1}}%
31   \def\bbl@trim@c{%
32     \ifx\bbl@trim@a\@sptoken
33       \expandafter\bbl@trim@b
34     \else
35       \expandafter\bbl@trim@b\expandafter#1%
36     \fi}%
37   \long\def\bbl@trim@b#1##1 \@nil{\bbl@trim@i##1}}
38 \bbl@tempa{ }
39 \long\def\bbl@trim@i#1\@nil#2\relax#3{#3{#1}}
40 \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 $\epsilon$-tex engine, it is based on \ifcsname, which is more efficient, and do not waste memory.

```
41 \def\bbl@ifunset#1{%
42   \expandafter\ifx\csname#1\endcsname\relax
43     \expandafter\@firstoftwo
44   \else
45     \expandafter\@secondoftwo
46   \fi}
47 \bbl@ifunset{ifcsname}%
48   {}%
49   {\def\bbl@ifunset#1{%
50     \ifcsname#1\endcsname
51       \expandafter\ifx\csname#1\endcsname\relax
52         \bbl@afterelse\expandafter\@firstoftwo
53       \else
54         \bbl@afterfi\expandafter\@secondoftwo
55       \fi
56     \else
57       \expandafter\@firstoftwo
58     \fi}}
```

\bbl@ifblank  A tool from url, by Donald Arseneau, which tests if a string is empty or space.

```
59 \def\bbl@ifblank#1{%
60   \bbl@ifblank@i#1\@nil\@nil\@secondoftwo\@firstoftwo\@nil}
61 \long\def\bbl@ifblank@i#1#2\@nil#3#4#5\@nil{#4}
```

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).

```
62 \def\bbl@forkv#1#2{%
63   \def\bbl@kvcmd##1##2##3{#2}%
64   \bbl@kvnext#1,\@nil,}
65 \def\bbl@kvnext#1,{%
```

```
66    \ifx\@nil#1\relax\else
67      \bbl@ifblank{#1}{}{\bbl@forkv@eq#1=\@empty=\@nil{#1}}%
68      \expandafter\bbl@kvnext
69    \fi}
70 \def\bbl@forkv@eq#1=#2=#3\@nil#4{%
71    \bbl@trim@def\bbl@forkv@a{#1}%
72    \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).

```
73 \def\bbl@vforeach#1#2{%
74    \def\bbl@forcmd##1{#2}%
75    \bbl@fornext#1,\@nil,}
76 \def\bbl@fornext#1,{%
77    \ifx\@nil#1\relax\else
78      \bbl@ifblank{#1}{}{\bbl@trim\bbl@forcmd{#1}}%
79      \expandafter\bbl@fornext
80    \fi}
81 \def\bbl@foreach#1{\expandafter\bbl@vforeach\expandafter{#1}}
```

\bbl@replace

```
82 \def\bbl@replace#1#2#3{% in #1 -> repl #2 by #3
83    \toks@{}%
84    \def\bbl@replace@aux##1#2##2#2{%
85      \ifx\bbl@nil##2%
86        \toks@\expandafter{\the\toks@##1}%
87      \else
88        \toks@\expandafter{\the\toks@##1#3}%
89        \bbl@afterfi
90        \bbl@replace@aux##2#2%
91      \fi}%
92    \expandafter\bbl@replace@aux#1#2\bbl@nil#2%
93    \edef#1{\the\toks@}}
```

\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.

```
94 \def\bbl@exp#1{%
95    \begingroup
96      \let\\\noexpand
97      \def\<##1>{\expandafter\noexpand\csname##1\endcsname}%
98      \edef\bbl@exp@aux{\endgroup#1}%
99    \bbl@exp@aux}
```

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.

```
100 \def\bbl@ifsamestring#1#2{%
101    \begingroup
102      \protected@edef\bbl@tempb{#1}%
103      \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
104      \protected@edef\bbl@tempc{#2}%
105      \edef\bbl@tempc{\expandafter\strip@prefix\meaning\bbl@tempc}%
106      \ifx\bbl@tempb\bbl@tempc
107        \aftergroup\@firstoftwo
108      \else
109        \aftergroup\@secondoftwo
```

```
110    \fi
111  \endgroup}
112 \chardef\bbl@engine=%
113  \ifx\directlua\@undefined
114    \ifx\XeTeXinputencoding\@undefined
115      \z@
116    \else
117      \tw@
118    \fi
119  \else
120    \@ne
121  \fi
```
122 ⟨⟨/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.

123 ⟨⟨∗Make sure ProvidesFile is defined⟩⟩ ≡
```
124 \ifx\ProvidesFile\@undefined
125  \def\ProvidesFile#1[#2 #3 #4]{%
126    \wlog{File: #1 #4 #3 <#2>}%
127    \let\ProvidesFile\@undefined}
128 \fi
```
129 ⟨⟨/Make sure ProvidesFile is defined⟩⟩

The following code is used in `babel.sty` and `babel.def`, and loads (only once) the data in `language.dat`.

130 ⟨⟨∗Load patterns in luatex⟩⟩ ≡
```
131 \ifx\directlua\@undefined\else
132  \ifx\bbl@luapatterns\@undefined
133    \input luababel.def
134  \fi
135 \fi
```
136 ⟨⟨/Load patterns in luatex⟩⟩

The following code is used in `babel.def` and `switch.def`.

137 ⟨⟨∗Load macros for plain if not LaTeX⟩⟩ ≡
```
138 \ifx\AtBeginDocument\@undefined
139  \input plain.def\relax
140 \fi
```
141 ⟨⟨/Load macros for plain if not LaTeX⟩⟩

## 6.1   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.

142 ⟨⟨∗Define core switching macros⟩⟩ ≡
```
143 \ifx\language\@undefined
144  \csname newcount\endcsname\language
145 \fi
```
146 ⟨⟨/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  To add languages to TEX's memory plain TEX version 3.0 supplies \newlanguage, in a pre-3.0 environment a similar macro has to be provided. For both cases a new macro is defined here, because the original \newlanguage was defined to be \outer.

For a format based on plain version 2.x, the definition of \newlanguage can not be copied because \count 19 is used for other purposes in these formats. Therefore \addlanguage is defined using a definition based on the macros used to define \newlanguage in plain TEX version 3.0.

For formats based on plain version 3.0 the definition of \newlanguage can be simply copied, removing \outer. Plain TEX version 3.0 uses \count 19 for this purpose.

```
147 ⟨⟨∗Define core switching macros⟩⟩ ≡
148 \ifx\newlanguage\@undefined
149   \csname newcount\endcsname\last@language
150   \def\addlanguage#1{%
151     \global\advance\last@language\@ne
152     \ifnum\last@language<\@cclvi
153     \else
154       \errmessage{No room for a new \string\language!}%
155     \fi
156     \global\chardef#1\last@language
157     \wlog{\string#1 = \string\language\the\last@language}}
158 \else
159   \countdef\last@language=19
160   \def\addlanguage{\alloc@9\language\chardef\@cclvi}
161 \fi
162 ⟨⟨/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 The Package File (LATEX, babel.sty)

In order to make use of the features of LATEX 2ε, the babel system contains a package file, babel.sty. This file is loaded by the \usepackage command and defines all the language options whose name is different from that of the .ldf file (like variant spellings). It 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.

## 7.1 base

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

```
163 ⟨∗package⟩
164 \NeedsTeXFormat{LaTeX2e}[2005/12/01]
165 \ProvidesPackage{babel}[⟨⟨date⟩⟩ ⟨⟨version⟩⟩ The Babel package]
166 \@ifpackagewith{babel}{debug}
167   {\providecommand\bbl@trace[1]{\message{^^J[ #1 ]}}%
168    \let\bbl@debug\@firstofone}
169   {\providecommand\bbl@trace[1]{}%
170    \let\bbl@debug\@gobble}
171 \ifx\bbl@switchflag\@undefined % Prevent double input
172   \let\bbl@switchflag\relax
173   \input switch.def\relax
174 \fi
175 ⟨⟨Load patterns in luatex⟩⟩
176 ⟨⟨Basic macros⟩⟩
177 \def\AfterBabelLanguage#1{%
178   \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.

```
179 \ifx\bbl@languages\@undefined\else
180   \begingroup
181     \catcode`\^^I=12
182     \@ifpackagewith{babel}{showlanguages}{%
183       \begingroup
184         \def\bbl@elt#1#2#3#4{\wlog{#2^^I#1^^I#3^^I#4}}%
185         \wlog{<*languages>}%
186         \bbl@languages
187         \wlog{</languages>}%
188       \endgroup}{}
189   \endgroup
190   \def\bbl@elt#1#2#3#4{%
191     \ifnum#2=\z@
192       \gdef\bbl@nulllanguage{#1}%
193       \def\bbl@elt##1##2##3##4{}%
194     \fi}%
195   \bbl@languages
196 \fi
197 \ifodd\bbl@engine
198   \def\bbl@loadbidi#1{%
199     \let\bbl@beforeforeign\leavevmode
200     \AtEndOfPackage{\EnableBabelHook{babel-bidi}}%
201     \RequirePackage{luatexbase}%
202     \directlua{
203       require('babel-bidi.lua')
204       require('babel-bidi-#1.lua')
205       luatexbase.add_to_callback('pre_linebreak_filter',
206         Babel.pre_otfload_v,
207         'Babel.pre_otfload_v',
208         luatexbase.priority_in_callback('pre_linebreak_filter',
209           'luaotfload.node_processor') or nil)
210       luatexbase.add_to_callback('hpack_filter',
211         Babel.pre_otfload_h,
212         'Babel.pre_otfload_h',
213         luatexbase.priority_in_callback('hpack_filter',
214           'luaotfload.node_processor') or nil)}}
215   \@ifpackagewith{babel}{bidi=basic-r}{% must go before any \DeclareOption
216     \bbl@loadbidi{basic-r}}{}
217   \@ifpackagewith{babel}{bidi=basic}{%
218     \bbl@loadbidi{basic}}{}
```

```
219 \fi
```

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. Useful for old versions of polyglossia, too.

```
220 \bbl@trace{Defining option 'base'}
221 \@ifpackagewith{babel}{base}{%
222   \ifx\directlua\@undefined
223     \DeclareOption*{\bbl@patterns{\CurrentOption}}%
224   \else
225     \DeclareOption*{\bbl@patterns@lua{\CurrentOption}}%
226   \fi
227   \DeclareOption{base}{}%
228   \DeclareOption{showlanguages}{}%
229   \ProcessOptions
230   \global\expandafter\let\csname opt@babel.sty\endcsname\relax
231   \global\expandafter\let\csname ver@babel.sty\endcsname\relax
232   \global\let\@ifl@ter@@\@ifl@ter
233   \def\@ifl@ter#1#2#3#4#5{\global\let\@ifl@ter\@ifl@ter@@}%
234   \endinput}{}%
```

## 7.2  `key=value` **options and other general option**

The following macros extract language modifiers, and only real package options are kept in the option list. Modifiers are saved and assigned to \BabelModifiers at \bbl@load@language; when no modifiers have been given, the former is \relax. How modifiers are handled are left to language styles; they can use \in@, loop them with \@for or load keyval, for example.

```
235 \bbl@trace{key=value and another general options}
236 \bbl@csarg\let{tempa\expandafter}\csname opt@babel.sty\endcsname
237 \def\bbl@tempb#1.#2{%
238   #1\ifx\@empty#2\else,\bbl@afterfi\bbl@tempb#2\fi}%
239 \def\bbl@tempd#1.#2\@nnil{%
240   \ifx\@empty#2%
241     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
242   \else
243     \in@{=}{#1}\ifin@
244       \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.#2}%
245     \else
246       \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
247       \bbl@csarg\edef{mod@#1}{\bbl@tempb#2}%
248     \fi
249   \fi}
250 \let\bbl@tempc\@empty
251 \bbl@foreach\bbl@tempa{\bbl@tempd#1.\@empty\@nnil}
252 \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.

```
253 \DeclareOption{KeepShorthandsActive}{}
254 \DeclareOption{activeacute}{}
255 \DeclareOption{activegrave}{}
256 \DeclareOption{debug}{}
257 \DeclareOption{noconfigs}{}
258 \DeclareOption{showlanguages}{}
259 \DeclareOption{silent}{}
```

```
260 \DeclareOption{shorthands=off}{\bbl@tempa shorthands=\bbl@tempa}
261 ⟨⟨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.

```
262 \let\bbl@opt@shorthands\@nnil
263 \let\bbl@opt@config\@nnil
264 \let\bbl@opt@main\@nnil
265 \let\bbl@opt@headfoot\@nnil
266 \let\bbl@opt@layout\@nnil
```

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

```
267 \def\bbl@tempa#1=#2\bbl@tempa{%
268   \bbl@csarg\ifx{opt@#1}\@nnil
269     \bbl@csarg\edef{opt@#1}{#2}%
270   \else
271     \bbl@error{%
272       Bad option `#1=#2'. Either you have misspelled the\\%
273       key or there is a previous setting of `#1'}{%
274       Valid keys are `shorthands', `config', `strings', `main',\\%
275       `headfoot', `safe', `math', among others.}
276   \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.

```
277 \let\bbl@language@opts\@empty
278 \DeclareOption*{%
279   \bbl@xin@{\string=}{\CurrentOption}%
280   \ifin@
281     \expandafter\bbl@tempa\CurrentOption\bbl@tempa
282   \else
283     \bbl@add@list\bbl@language@opts{\CurrentOption}%
284   \fi}
```

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

```
285 \ProcessOptions*
```

## 7.3   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@ifshorthands is always true, and it is always false if shorthands is empty. Also, some code makes sense only with shorthands=....

```
286 \bbl@trace{Conditional loading of shorthands}
287 \def\bbl@sh@string#1{%
288   \ifx#1\@empty\else
289     \ifx#1t\string~%
290     \else\ifx#1c\string,%
291     \else\string#1%
292     \fi\fi
293     \expandafter\bbl@sh@string
```

```
294    \fi}
295 \ifx\bbl@opt@shorthands\@nnil
296    \def\bbl@ifshorthand#1#2#3{#2}%
297 \else\ifx\bbl@opt@shorthands\@empty
298    \def\bbl@ifshorthand#1#2#3{#3}%
299 \else
```

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

```
300    \def\bbl@ifshorthand#1{%
301      \bbl@xin@{\string#1}{\bbl@opt@shorthands}%
302      \ifin@
303        \expandafter\@firstoftwo
304      \else
305        \expandafter\@secondoftwo
306      \fi}
```

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

```
307    \edef\bbl@opt@shorthands{%
308      \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.

```
309    \bbl@ifshorthand{'}%
310      {\PassOptionsToPackage{activeacute}{babel}}{}
311    \bbl@ifshorthand{`}%
312      {\PassOptionsToPackage{activegrave}{babel}}{}
313 \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.

```
314 \ifx\bbl@opt@headfoot\@nnil\else
315    \g@addto@macro\@resetactivechars{%
316      \set@typeset@protect
317      \expandafter\select@language@x\expandafter{\bbl@opt@headfoot}%
318      \let\protect\noexpand}
319 \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.

```
320 \ifx\bbl@opt@safe\@undefined
321    \def\bbl@opt@safe{BR}
322 \fi
323 \ifx\bbl@opt@main\@nnil\else
324    \edef\bbl@language@opts{%
325      \ifx\bbl@language@opts\@empty\else\bbl@language@opts,\fi
326        \bbl@opt@main}
327 \fi
```

For layout an auxiliary macro is provided, available for packages and language styles.

```
328 \bbl@trace{Defining IfBabelLayout}
329 \ifx\bbl@opt@layout\@nnil
330    \newcommand\IfBabelLayout[3]{#3}%
331 \else
332    \newcommand\IfBabelLayout[1]{%
333      \@expandtwoargs\in@{.#1.}{.\bbl@opt@layout.}%
334      \ifin@
335        \expandafter\@firstoftwo
```

```
336    \else
337      \expandafter\@secondoftwo
338    \fi}
339 \fi
```

## 7.4 Language options

Languages are loaded when processing the corresponding option *except* if a main language
has been set. In such a case, it is not loaded until all options has been processed. The
following macro inputs the ldf file and does some additional checks (\input works, too,
but possible errors are not catched).

```
340 \bbl@trace{Language options}
341 \let\bbl@afterlang\relax
342 \let\BabelModifiers\relax
343 \let\bbl@loaded\@empty
344 \def\bbl@load@language#1{%
345   \InputIfFileExists{#1.ldf}%
346     {\edef\bbl@loaded{\CurrentOption
347       \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}%
348     \expandafter\let\expandafter\bbl@afterlang
349       \csname\CurrentOption.ldf-h@@k\endcsname
350     \expandafter\let\expandafter\BabelModifiers
351       \csname bbl@mod@\CurrentOption\endcsname}%
352     {\bbl@error{%
353       Unknown option `\CurrentOption'. Either you misspelled it\\%
354       or the language definition file \CurrentOption.ldf was not found}{%
355       Valid options are: shorthands=, KeepShorthandsActive,\\%
356       activeacute, activegrave, noconfigs, safe=, main=, math=\\%
357       headfoot=, strings=, config=, hyphenmap=, or a language name.}}}
```

Now, we set language options whose names are different from ldf files.

```
358 \def\bbl@try@load@lang#1#2#3{%
359   \IfFileExists{\CurrentOption.ldf}%
360     {\bbl@load@language{\CurrentOption}}%
361     {#1\bbl@load@language{#2}#3}}
362 \DeclareOption{afrikaans}{\bbl@try@load@lang{}{dutch}{}}
363 \DeclareOption{brazil}{\bbl@try@load@lang{}{portuges}{}}
364 \DeclareOption{brazilian}{\bbl@try@load@lang{}{portuges}{}}
365 \DeclareOption{hebrew}{%
366   \input{rlbabel.def}%
367   \bbl@load@language{hebrew}}
368 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}{}}
369 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}{}}
370 \DeclareOption{nynorsk}{\bbl@try@load@lang{}{norsk}{}}
371 \DeclareOption{polutonikogreek}{%
372   \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}
373 \DeclareOption{portuguese}{\bbl@try@load@lang{}{portuges}{}}
374 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}{}}
375 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}{}}
376 \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.

```
377 \ifx\bbl@opt@config\@nnil
```

```
378    \@ifpackagewith{babel}{noconfigs}{}%
379      {\InputIfFileExists{bblopts.cfg}%
380        {\typeout{************************************^^J%
381                  * Local config file bblopts.cfg used^^J%
382                  *}}%
383        {}}%
384 \else
385    \InputIfFileExists{\bbl@opt@config.cfg}%
386      {\typeout{************************************^^J%
387                * Local config file \bbl@opt@config.cfg used^^J%
388                *}}%
389      {\bbl@error{%
390         Local config file `\bbl@opt@config.cfg' not found}{%
391         Perhaps you misspelled it.}}%
392 \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 name of the option and the file are the same.

```
393 \bbl@for\bbl@tempa\bbl@language@opts{%
394    \bbl@ifunset{ds@\bbl@tempa}%
395      {\edef\bbl@tempb{%
396         \noexpand\DeclareOption
397           {\bbl@tempa}%
398           {\noexpand\bbl@load@language{\bbl@tempa}}}%
399      \bbl@tempb}%
400      \@empty}
```

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 accesing the file system just to see if the option could be a language.

```
401 \bbl@foreach\@classoptionslist{%
402    \bbl@ifunset{ds@#1}%
403      {\IfFileExists{#1.ldf}%
404        {\DeclareOption{#1}{\bbl@load@language{#1}}}%
405        {}}%
406      {}}
```

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

```
407 \ifx\bbl@opt@main\@nnil\else
408    \expandafter
409    \let\expandafter\bbl@loadmain\csname ds@\bbl@opt@main\endcsname
410    \DeclareOption{\bbl@opt@main}{}
411 \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):

```
412 \def\AfterBabelLanguage#1{%
413    \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang}{}}
414 \DeclareOption*{}
415 \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.

```
416 \ifx\bbl@opt@main\@nnil
417   \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}
418   \let\bbl@tempc\@empty
419   \bbl@for\bbl@tempb\bbl@tempa{%
420     \bbl@xin@{,\bbl@tempb,}{,\bbl@loaded,}%
421     \ifin@\edef\bbl@tempc{\bbl@tempb}\fi}
422   \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
423   \expandafter\bbl@tempa\bbl@loaded,\@nnil
424   \ifx\bbl@tempb\bbl@tempc\else
425     \bbl@warning{%
426       Last declared language option is `\bbl@tempc',\\%
427       but the last processed one was `\bbl@tempb'.\\%
428       The main language cannot be set as both a global\\%
429       and a package option. Use `main=\bbl@tempc' as\\%
430       option. Reported}%
431   \fi
432 \else
433   \DeclareOption{\bbl@opt@main}{\bbl@loadmain}
434   \ExecuteOptions{\bbl@opt@main}
435   \DeclareOption*{}
436   \ProcessOptions*
437 \fi
438 \def\AfterBabelLanguage{%
439   \bbl@error
440     {Too late for \string\AfterBabelLanguage}%
441     {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.

```
442 \ifx\bbl@main@language\@undefined
443   \bbl@error{%
444     You haven't specified a language option}{%
445     You need to specify a language, either as a global option\\%
446     or as an optional argument to the \string\usepackage\space
447     command;\\%
448     You shouldn't try to proceed from here, type x to quit.}
449 \fi
450 ⟨/package⟩
451 ⟨*core⟩
```

# 8   The kernel of Babel (`babel.def`, common)

The kernel of the babel system is stored in either hyphen.cfg or switch.def and
babel.def. The file babel.def contains most of the code, while switch.def defines the
language switching commands; both can be read at run time. 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 (by default, it also inputs switch.def, for "historical reasons", but it
is not necessary). When babel.def is loaded it checks if the current version of switch.def
is in the format; if not, it is loaded. A further file, babel.sty, contains LATEX-specific stuff.
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

```
452 \ifx\ldf@quit\@undefined
453 \else
454   \expandafter\endinput
455 \fi
456 ⟨⟨Make sure ProvidesFile is defined⟩⟩
457 \ProvidesFile{babel.def}[⟨⟨date⟩⟩ ⟨⟨version⟩⟩ Babel common definitions]
458 ⟨⟨Load macros for plain if not LaTeX⟩⟩
```

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 in 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).

```
459 \ifx\bbl@ifshorthand\@undefined
460   \let\bbl@opt@shorthands\@nnil
461   \def\bbl@ifshorthand#1#2#3{#2}%
462   \let\bbl@language@opts\@empty
463   \ifx\babeloptionstrings\@undefined
464     \let\bbl@opt@strings\@nnil
465   \else
466     \let\bbl@opt@strings\babeloptionstrings
467   \fi
468   \def\BabelStringsDefault{generic}
469   \def\bbl@tempa{normal}
470   \ifx\babeloptionmath\bbl@tempa
471     \def\bbl@mathnormal{\noexpand\textormath}
472   \fi
473   \def\AfterBabelLanguage#1#2{}
474   \ifx\BabelModifiers\@undefined\let\BabelModifiers\relax\fi
475   \let\bbl@afterlang\relax
476   \def\bbl@opt@safe{BR}
477   \ifx\@uclclist\@undefined\let\@uclclist\@empty\fi
478   \ifx\bbl@trace\@undefined\def\bbl@trace#1{}\fi
479 \fi
```

And continue.

```
480 \ifx\bbl@switchflag\@undefined % Prevent double input
481   \let\bbl@switchflag\relax
482   \input switch.def\relax
483 \fi
484 \bbl@trace{Compatibility with language.def}
485 \ifx\bbl@languages\@undefined
486   \ifx\directlua\@undefined
487     \openin1 = language.def
488     \ifeof1
489       \closein1
490       \message{I couldn't find the file language.def}
491     \else
492       \closein1
493       \begingroup
494         \def\addlanguage#1#2#3#4#5{%
495           \expandafter\ifx\csname lang@#1\endcsname\relax\else
```

```
496          \global\expandafter\let\csname l@#1\expandafter\endcsname
497            \csname lang@#1\endcsname
498        \fi}%
499      \def\uselanguage#1{}%
500      \input language.def
501    \endgroup
502  \fi
503  \fi
504  \chardef\l@english\z@
505 \fi
506 ⟨⟨Load patterns in luatex⟩⟩
507 ⟨⟨Basic macros⟩⟩
```

\addto  For each language four control sequences have to be defined that control the
language-specific definitions. To be able to add something to these macro once they have
been defined the macro \addto is introduced. It takes two arguments, a ⟨control sequence⟩
and TeX-code to be added to the ⟨control sequence⟩.

If the ⟨control sequence⟩ has not been defined before it is defined now. The control
sequence could also expand to \relax, in which case a circular definition results. The net
result is a stack overflow. Otherwise the replacement text for the ⟨control sequence⟩ is
expanded and stored in a token register, together with the TeX-code to be added. Finally
the ⟨control sequence⟩ is redefined, using the contents of the token register.

```
508 \def\addto#1#2{%
509   \ifx#1\@undefined
510     \def#1{#2}%
511   \else
512     \ifx#1\relax
513       \def#1{#2}%
514     \else
515       {\toks@\expandafter{#1#2}%
516         \xdef#1{\the\toks@}}%
517     \fi
518   \fi}
```

The macro \initiate@active@char takes all the necessary actions to make its argument a
shorthand character. The real work is performed once for each character.

```
519 \def\bbl@withactive#1#2{%
520   \begingroup
521     \lccode`~=`#2\relax
522     \lowercase{\endgroup#1~}}
```

\bbl@redefine  To redefine a command, we save the old meaning of the macro. Then we redefine it to call
the original macro with the 'sanitized' argument. The reason why we do it this way is that
we don't want to redefine the LaTeX macros completely in case their definitions change
(they have changed in the past).

Because we need to redefine a number of commands we define the command
\bbl@redefine which takes care of this. It creates a new control sequence, \org@...

```
523 \def\bbl@redefine#1{%
524   \edef\bbl@tempa{\bbl@stripslash#1}%
525   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
526   \expandafter\def\csname\bbl@tempa\endcsname}
```

This command should only be used in the preamble of the document.

```
527 \@onlypreamble\bbl@redefine
```

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

```
528 \def\bbl@redefine@long#1{%
529   \edef\bbl@tempa{\bbl@stripslash#1}%
530   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
531   \expandafter\long\expandafter\def\csname\bbl@tempa\endcsname}
532 \@onlypreamble\bbl@redefine@long
```

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␣.

```
533 \def\bbl@redefinerobust#1{%
534   \edef\bbl@tempa{\bbl@stripslash#1}%
535   \bbl@ifunset{\bbl@tempa\space}%
536     {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
537      \bbl@exp{\def\\#1{\\\protect\<\bbl@tempa\space>}}}%
538     {\bbl@exp{\let\<org@\bbl@tempa>\<\bbl@tempa\space>}}%
539     \@namedef{\bbl@tempa\space}}
```

This command should only be used in the preamble of the document.

```
540 \@onlypreamble\bbl@redefinerobust
```

## 8.2 Hooks

Note they are loaded in babel.def. switch.def only provides a "hook" for hooks (with a default value which is a no-op, below). Admittedly, the current implementation is a somewhat simplistic and does vety little to catch errors, but it is intended for developpers, after all. \bbl@usehooks is the commands used by babel to execute hooks defined for an event.

```
541 \bbl@trace{Hooks}
542 \def\AddBabelHook#1#2{%
543   \bbl@ifunset{bbl@hk@#1}{\EnableBabelHook{#1}}{}%
544   \def\bbl@tempa##1,#2=##2,##3\@empty{\def\bbl@tempb{##2}}%
545   \expandafter\bbl@tempa\bbl@evargs,#2=,\@empty
546   \bbl@ifunset{bbl@ev@#1@#2}%
547     {\bbl@csarg\bbl@add{ev@#2}{\bbl@elt{#1}}%
548      \bbl@csarg\newcommand}%
549     {\bbl@csarg\let{ev@#1@#2}\relax
550      \bbl@csarg\newcommand}%
551   {ev@#1@#2}[\bbl@tempb]}
552 \def\EnableBabelHook#1{\bbl@csarg\let{hk@#1}\@firstofone}
553 \def\DisableBabelHook#1{\bbl@csarg\let{hk@#1}\@gobble}
554 \def\bbl@usehooks#1#2{%
555   \def\bbl@elt##1{%
556     \@nameuse{bbl@hk@##1}{\@nameuse{bbl@ev@##1@#1}#2}}%
557   \@nameuse{bbl@ev@#1}}
```

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).

```
558 \def\bbl@evargs{,% don't delete the comma
559   everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
560   adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
561   beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
562   hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0}
```

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.

```
563 \bbl@trace{Defining babelensure}
564 \newcommand\babelensure[2][]{%  TODO - revise test files
565   \AddBabelHook{babel-ensure}{afterextras}{%
566     \ifcase\bbl@select@type
567       \@nameuse{bbl@e@\languagename}%
568     \fi}%
569   \begingroup
570     \let\bbl@ens@include\@empty
571     \let\bbl@ens@exclude\@empty
572     \def\bbl@ens@fontenc{\relax}%
573     \def\bbl@tempb##1{%
574       \ifx\@empty##1\else\noexpand##1\expandafter\bbl@tempb\fi}%
575     \edef\bbl@tempa{\bbl@tempb#1\@empty}%
576     \def\bbl@tempb##1=##2\@@{\@namedef{bbl@ens@##1}{##2}}%
577     \bbl@foreach\bbl@tempa{\bbl@tempb##1\@@}%
578     \def\bbl@tempc{\bbl@ensure}%
579     \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
580       \expandafter{\bbl@ens@include}}%
581     \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
582       \expandafter{\bbl@ens@exclude}}%
583     \toks@\expandafter{\bbl@tempc}%
584     \bbl@exp{%
585   \endgroup
586   \def\<bbl@e@#2>{\the\toks@{\bbl@ens@fontenc}}}}}
587 \def\bbl@ensure#1#2#3{% 1: include 2: exclude 3: fontenc
588   \def\bbl@tempb##1{% elt for (excluding) \bbl@captionslist list
589     \ifx##1\@empty\else
590       \in@{##1}{#2}%
591       \ifin@\else
592         \bbl@ifunset{bbl@ensure@\languagename}%
593           {\bbl@exp{%
594             \\\DeclareRobustCommand\<bbl@ensure@\languagename>[1]{%
595               \\\foreignlanguage{\languagename}%
596               {\ifx\relax#3\else
597                 \\\fontencoding{#3}\\\selectfont
598               \fi
599               ########1}}}}%
600           {}%
601         \toks@\expandafter{##1}%
602         \edef##1{%
603           \bbl@csarg\noexpand{ensure@\languagename}%
604           {\the\toks@}}%
605       \fi
606       \expandafter\bbl@tempb
607     \fi}%
608   \expandafter\bbl@tempb\bbl@captionslist\today\@empty
609   \def\bbl@tempa##1{% elt for include list
610     \ifx##1\@empty\else
611       \bbl@csarg\in@{ensure@\languagename\expandafter}\expandafter{##1}%
```

64

```
612       \ifin@\else
613         \bbl@tempb##1\@empty
614       \fi
615       \expandafter\bbl@tempa
616     \fi}%
617   \bbl@tempa#1\@empty}
618 \def\bbl@captionslist{%
619   \prefacename\refname\abstractname\bibname\chaptername\appendixname
620   \contentsname\listfigurename\listtablename\indexname\figurename
621   \tablename\partname\enclname\ccname\headtoname\pagename\seename
622   \alsoname\proofname\glossaryname}
```

## 8.3   Setting up language files

\LdfInit  The second version of \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.

```
623 \bbl@trace{Macros for setting language files up}
624 \def\bbl@ldfinit{%
625   \let\bbl@screset\@empty
626   \let\BabelStrings\bbl@opt@string
627   \let\BabelOptions\@empty
628   \let\BabelLanguages\relax
629   \ifx\originalTeX\@undefined
630     \let\originalTeX\@empty
631   \else
632     \originalTeX
633   \fi}
634 \def\LdfInit#1#2{%
635   \chardef\atcatcode=\catcode`\@
636   \catcode`\@=11\relax
637   \chardef\eqcatcode=\catcode`\=
638   \catcode`\==12\relax
639   \expandafter\if\expandafter\@backslashchar
640                 \expandafter\@car\string#2\@nil
641     \ifx#2\@undefined\else
642       \ldf@quit{#1}%
643     \fi
644   \else
```

```
645        \expandafter\ifx\csname#2\endcsname\relax\else
646          \ldf@quit{#1}%
647        \fi
648      \fi
649    \bbl@ldfinit}
```

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

```
650 \def\ldf@quit#1{%
651   \expandafter\main@language\expandafter{#1}%
652   \catcode`\@=\atcatcode \let\atcatcode\relax
653   \catcode`\==\eqcatcode \let\eqcatcode\relax
654   \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.

```
655 \def\bbl@afterldf#1{%
656   \bbl@afterlang
657   \let\bbl@afterlang\relax
658   \let\BabelModifiers\relax
659   \let\bbl@screset\relax}%
660 \def\ldf@finish#1{%
661   \loadlocalcfg{#1}%
662   \bbl@afterldf{#1}%
663   \expandafter\main@language\expandafter{#1}%
664   \catcode`\@=\atcatcode \let\atcatcode\relax
665   \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.

```
666 \@onlypreamble\LdfInit
667 \@onlypreamble\ldf@quit
668 \@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.

```
669 \def\main@language#1{%
670   \def\bbl@main@language{#1}%
671   \let\languagename\bbl@main@language
672   \bbl@patterns{\languagename}}
```

We also have to make sure that some code gets executed at the beginning of the document.
Languages does not set \pagedir, so we set here for the whole document to the main
\bodydir.

```
673 \AtBeginDocument{%
674   \expandafter\selectlanguage\expandafter{\bbl@main@language}%
675   \ifcase\bbl@engine\or\pagedir\bodydir\fi}  % TODO - a better place
```

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

```
676 \def\select@language@x#1{%
677   \ifcase\bbl@select@type
678     \bbl@ifsamestring\languagename{#1}{}{\select@language{#1}}%
679   \else
680     \select@language{#1}%
681   \fi}
```

### 8.4 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.

```
682 \bbl@trace{Shorhands}
683 \def\bbl@add@special#1{% 1:a macro like \", \?, etc.
684   \bbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
685   \bbl@ifunset{@sanitize}{}{\bbl@add\@sanitize{\@makeother#1}}%
686   \ifx\nfss@catcodes\@undefined\else % TODO - same for above
687     \begingroup
688       \catcode`#1\active
689       \nfss@catcodes
690       \ifnum\catcode`#1=\active
691         \endgroup
692         \bbl@add\nfss@catcodes{\@makeother#1}%
693       \else
694         \endgroup
695       \fi
696   \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.

```
697 \def\bbl@remove@special#1{%
698   \begingroup
699     \def\x##1##2{\ifnum`#1=`##2\noexpand\@empty
700                 \else\noexpand##1\noexpand##2\fi}%
701     \def\do{\x\do}%
702     \def\@makeother{\x\@makeother}%
703   \edef\x{\endgroup
704     \def\noexpand\dospecials{\dospecials}%
705     \expandafter\ifx\csname @sanitize\endcsname\relax\else
706       \def\noexpand\@sanitize{\@sanitize}%
707     \fi}%
708   \x}
```

\initiate@active@char    A language definition file can call this macro to make a character active. This macro takes
one argument, the character that is to be made active. When the character was already
active this macro does nothing. Otherwise, this macro defines the control sequence
\normal@char⟨char⟩ to expand to the character in its 'normal state' and it defines the
active character to expand to \normal@char⟨char⟩ by default (⟨char⟩ being the character
to be made active). Later its definition can be changed to expand to \active@char⟨char⟩
by calling \bbl@activate{⟨char⟩}.

For example, to make the double quote character active one could have
\initiate@active@char{"} in a language definition file. This defines " as
\active@prefix "\active@char" (where the first " is the character with its original
catcode, when the shorthand is created, and \active@char" is a single token). In protected
contexts, it expands to \protect " or \noexpand " (ie, with the original "); otherwise
\active@char" is executed. This macro in turn expands to \normal@char" in "safe"
contexts (eg, \label), but \user@active" in normal "unsafe" ones. The latter search a
definition in the user, language and system levels, in this order, but if none is found,
\normal@char" is used. However, a deactivated shorthand (with \bbl@deactivate is
defined as \active@prefix "\normal@char".

The following macro is used to define shorthands in the three levels. It takes 4 arguments: the (string'ed) character, \<level>@group, <level>@active and <next-level>@active (except in system).

```
709 \def\bbl@active@def#1#2#3#4{%
710   \@namedef{#3#1}{%
711     \expandafter\ifx\csname#2@sh@#1@\endcsname\relax
712       \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
713     \else
714       \bbl@afterfi\csname#2@sh@#1@\endcsname
715     \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.

```
716   \long\@namedef{#3@arg#1}##1{%
717     \expandafter\ifx\csname#2@sh@#1@\string##1@\endcsname\relax
718       \bbl@afterelse\csname#4#1\endcsname##1%
719     \else
720       \bbl@afterfi\csname#2@sh@#1@\string##1@\endcsname
721     \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.

```
722 \def\initiate@active@char#1{%
723   \bbl@ifunset{active@char\string#1}%
724     {\bbl@withactive
725       {\expandafter\@initiate@active@char\expandafter}#1\string#1#1}%
726     {}}
```

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).

```
727 \def\@initiate@active@char#1#2#3{%
728   \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
729   \ifx#1\@undefined
730     \bbl@csarg\edef{oridef@#2}{\let\noexpand#1\noexpand\@undefined}%
731   \else
732     \bbl@csarg\let{oridef@@#2}#1%
733     \bbl@csarg\edef{oridef@#2}{%
734       \let\noexpand#1%
735       \expandafter\noexpand\csname bbl@oridef@@#2\endcsname}%
736   \fi
```

If the character is already active we provide the default expansion under this shorthand mechanism. Otherwise we write a message in the transcript file, and define \normal@char⟨char⟩ to expand to the character in its default state. If the character is mathematically active when babel is loaded (for example ') the normal expansion is somewhat different to avoid an infinite loop (but it does not prevent the loop if the mathcode is set to "8000 *a posteriori*).

```
737   \ifx#1#3\relax
738     \expandafter\let\csname normal@char#2\endcsname#3%
739   \else
740     \bbl@info{Making #2 an active character}%
741     \ifnum\mathcode`#2="8000
742       \@namedef{normal@char#2}{%
743         \textormath{#3}{\csname bbl@oridef@@#2\endcsname}}%
744     \else
745       \@namedef{normal@char#2}{#3}%
```

```
746        \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).

```
747        \bbl@restoreactive{#2}%
748        \AtBeginDocument{%
749          \catcode`#2\active
750          \if@filesw
751            \immediate\write\@mainaux{\catcode`\string#2\active}%
752          \fi}%
753        \expandafter\bbl@add@special\csname#2\endcsname
754        \catcode`#2\active
755      \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*⟩).

```
756    \let\bbl@tempa\@firstoftwo
757    \if\string^#2%
758      \def\bbl@tempa{\noexpand\textormath}%
759    \else
760      \ifx\bbl@mathnormal\@undefined\else
761        \let\bbl@tempa\bbl@mathnormal
762      \fi
763    \fi
764    \expandafter\edef\csname active@char#2\endcsname{%
765      \bbl@tempa
766        {\noexpand\if@safe@actives
767           \noexpand\expandafter
768           \expandafter\noexpand\csname normal@char#2\endcsname
769         \noexpand\else
770           \noexpand\expandafter
771           \expandafter\noexpand\csname bbl@doactive#2\endcsname
772         \noexpand\fi}%
773      {\expandafter\noexpand\csname normal@char#2\endcsname}}%
774    \bbl@csarg\edef{doactive#2}{%
775      \expandafter\noexpand\csname user@active#2\endcsname}%
```

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

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

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

```
776    \bbl@csarg\edef{active@#2}{%
777      \noexpand\active@prefix\noexpand#1%
778      \expandafter\noexpand\csname active@char#2\endcsname}%
779    \bbl@csarg\edef{normal@#2}{%
780      \noexpand\active@prefix\noexpand#1%
781      \expandafter\noexpand\csname normal@char#2\endcsname}%
782    \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.

```
783    \bbl@active@def#2\user@group{user@active}{language@active}%
784    \bbl@active@def#2\language@group{language@active}{system@active}%
785    \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.

```
786    \expandafter\edef\csname\user@group @sh@#2@@\endcsname
787      {\expandafter\noexpand\csname normal@char#2\endcsname}%
788    \expandafter\edef\csname\user@group @sh@#2@\string\protect@\endcsname
789      {\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.

```
790    \if\string'#2%
791      \let\prim@s\bbl@prim@s
792      \let\active@math@prime#1%
793    \fi
794    \bbl@usehooks{initiateactive}{{#1}{#2}{#3}}}
```

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

```
795 ⟨*More package options⟩ ≡
796 \DeclareOption{math=active}{}
797 \DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
798 ⟨/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.

```
799 \@ifpackagewith{babel}{KeepShorthandsActive}%
800   {\let\bbl@restoreactive\@gobble}%
801   {\def\bbl@restoreactive#1{%
802      \bbl@exp{%
803        \\\AfterBabelLanguage\\\CurrentOption
804          {\catcode`#1=\the\catcode`#1\relax}%
805        \\\AtEndOfPackage
806          {\catcode`#1=\the\catcode`#1\relax}}}%
807   \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.

```
808 \def\bbl@sh@select#1#2{%
809   \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
810      \bbl@afterelse\bbl@scndcs
811   \else
```

812     `\bbl@afterfi\csname#1@sh@#2@sel\endcsname`

813  `\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 `\protect`s the active character whenever `\protect` is *not* `\@typeset@protect`.

814 `\def\active@prefix#1{%`

815  `\ifx\protect\@typeset@protect`

816  `\else`

When `\protect` is set to `\@unexpandable@protect` we make sure that the active character is als *not* expanded by inserting `\noexpand` in front of it. The `\@gobble` is needed to remove a token such as `\activechar:` (when the double colon was the active character to be dealt with).

817    `\ifx\protect\@unexpandable@protect`

818     `\noexpand#1%`

819    `\else`

820     `\protect#1%`

821    `\fi`

822    `\expandafter\@gobble`

823  `\fi}`

**\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*⟩.

824 `\newif\if@safe@actives`

825 `\@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.

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

**\bbl@activate**    Both macros take one argument, like `\initiate@active@char`. The macro is used to
**\bbl@deactivate**    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`.

827 `\def\bbl@activate#1{%`

828  `\bbl@withactive{\expandafter\let\expandafter}#1%`

829    `\csname bbl@active@\string#1\endcsname}`

830 `\def\bbl@deactivate#1{%`

831  `\bbl@withactive{\expandafter\let\expandafter}#1%`

832    `\csname bbl@normal@\string#1\endcsname}`

**\bbl@firstcs**    These macros have two arguments. They use one of their arguments to build a control
**\bbl@scndcs**    sequence from.

833 `\def\bbl@firstcs#1#2{\csname#1\endcsname}`

834 `\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.

```
835 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
836 \def\@decl@short#1#2#3\@nil#4{%
837   \def\bbl@tempa{#3}%
838   \ifx\bbl@tempa\@empty
839     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
840     \bbl@ifunset{#1@sh@\string#2@}{}%
841       {\def\bbl@tempa{#4}%
842        \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbl@tempa
843        \else
844          \bbl@info
845            {Redefining #1 shorthand \string#2\\%
846             in language \CurrentOption}%
847        \fi}%
848     \@namedef{#1@sh@\string#2@}{#4}%
849   \else
850     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs
851     \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%
852       {\def\bbl@tempa{#4}%
853        \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
854        \else
855          \bbl@info
856            {Redefining #1 shorthand \string#2\string#3\\%
857             in language \CurrentOption}%
858        \fi}%
859     \@namedef{#1@sh@\string#2@\string#3@}{#4}%
860   \fi}
```

\textormath  Some of the shorthands that will be declared by the language definition files have to be
usable in both text and mathmode. To achieve this the helper macro \textormath is
provided.

```
861 \def\textormath{%
862   \ifmmode
863     \expandafter\@secondoftwo
864   \else
865     \expandafter\@firstoftwo
866   \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'.

```
867 \def\user@group{user}
868 \def\language@group{english}
869 \def\system@group{system}
```

\useshorthands  This is the user level command to tell LaTeX that user level shorthands will be used in the
document. It takes one argument, the character that starts a shorthand. First note that this
is user level, and then initialize and activate 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.

```
870 \def\useshorthands{%
871   \@ifstar\bbl@usesh@s{\bbl@usesh@x{}}}
872 \def\bbl@usesh@s#1{%
873   \bbl@usesh@x
874     {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bbl@activate{#1}}}%
875     {#1}}
876 \def\bbl@usesh@x#1#2{%
877   \bbl@ifshorthand{#2}%
```

```
878    {\def\user@group{user}%
879     \initiate@active@char{#2}%
880     #1%
881     \bbl@activate{#2}}%
882    {\bbl@error
883       {Cannot declare a shorthand turned off (\string#2)}
884       {Sorry, but you cannot use shorthands which have been\\%
885        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.

```
886 \def\user@language@group{user@\language@group}
887 \def\bbl@set@user@generic#1#2{%
888   \bbl@ifunset{user@generic@active#1}%
889     {\bbl@active@def#1\user@language@group{user@active}{user@generic@active}%
890      \bbl@active@def#1\user@group{user@generic@active}{language@active}%
891      \expandafter\edef\csname#2@sh@#1@@\endcsname{%
892        \expandafter\noexpand\csname normal@char#1\endcsname}%
893      \expandafter\edef\csname#2@sh@#1@\string\protect@\endcsname{%
894        \expandafter\noexpand\csname user@active#1\endcsname}}%
895   \@empty}
896 \newcommand\defineshorthand[3][user]{%
897   \edef\bbl@tempa{\zap@space#1 \@empty}%
898   \bbl@for\bbl@tempb\bbl@tempa{%
899     \if*\expandafter\@car\bbl@tempb\@nil
900       \edef\bbl@tempb{user@\expandafter\@gobble\bbl@tempb}%
901       \@expandtwoargs
902         \bbl@set@user@generic{\expandafter\string\@car#2\@nil}\bbl@tempb
903     \fi
904     \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.

```
905 \def\languageshorthands#1{\def\language@group{#1}}
```

\aliasshorthand  First the new shorthand needs to be initialized,

```
906 \def\aliasshorthand#1#2{%
907   \bbl@ifshorthand{#2}%
908     {\expandafter\ifx\csname active@char\string#2\endcsname\relax
909        \ifx\document\@notprerr
910          \@notshorthand{#2}%
911        \else
912          \initiate@active@char{#2}%
```

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".

```
913        \expandafter\let\csname active@char\string#2\expandafter\endcsname
914          \csname active@char\string#1\endcsname
915        \expandafter\let\csname normal@char\string#2\expandafter\endcsname
916          \csname normal@char\string#1\endcsname
917        \bbl@activate{#2}%
918      \fi
```

73

```
919        \fi}%
920      {\bbl@error
921        {Cannot declare a shorthand turned off (\string#2)}
922        {Sorry, but you cannot use shorthands which have been\\%
923         turned off in the package options}}}
```

```
924 \def\@notshorthand#1{%
925   \bbl@error{%
926     The character `\string #1' should be made a shorthand character;\\%
927     add the command \string\useshorthands\string{#1\string} to
928     the preamble.\\%
929     I will ignore your instruction}%
930    {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.

```
931 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
932 \DeclareRobustCommand*\shorthandoff{%
933   \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
934 \def\bbl@shorthandoff#1#2{\bbl@switch@sh#1#2\@nnil}
```

\bbl@switch@sh  The macro \bbl@switch@sh takes the list of characters apart one by one and subsequently
switches the category code of the shorthand character according to the first argument of
\bbl@switch@sh.
But before any of this switching takes place we make sure that the character we are
dealing with is known as a shorthand character. If it is, a macro such as \active@char"
should exist.
Switching off and on is easy – we just set the category code to 'other' (12) and \active.
With the starred version, the original catcode and the original definition, saved in
@initiate@active@char, are restored.

```
935 \def\bbl@switch@sh#1#2{%
936   \ifx#2\@nnil\else
937     \bbl@ifunset{bbl@active@\string#2}%
938       {\bbl@error
939         {I cannot switch `\string#2' on or off--not a shorthand}%
940         {This character is not a shorthand. Maybe you made\\%
941          a typing mistake? I will ignore your instruction}}%
942      {\ifcase#1%
943         \catcode`#212\relax
944       \or
945         \catcode`#2\active
946       \or
947         \csname bbl@oricat@\string#2\endcsname
948         \csname bbl@oridef@\string#2\endcsname
949       \fi}%
950     \bbl@afterfi\bbl@switch@sh#1%
951   \fi}
```

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

```
952 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
953 \def\bbl@putsh#1{%
954   \bbl@ifunset{bbl@active@\string#1}%
955     {\bbl@putsh@i#1\@empty\@nnil}%
956     {\csname bbl@active@\string#1\endcsname}}
957 \def\bbl@putsh@i#1#2\@nnil{%
```

```
958    \csname\languagename @sh@\string#1@%
959      \ifx\@empty#2\else\string#2@\fi\endcsname}
960 \ifx\bbl@opt@shorthands\@nnil\else
961    \let\bbl@s@initiate@active@char\initiate@active@char
962    \def\initiate@active@char#1{%
963      \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
964    \let\bbl@s@switch@sh\bbl@switch@sh
965    \def\bbl@switch@sh#1#2{%
966      \ifx#2\@nnil\else
967        \bbl@afterfi
968        \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
969      \fi}
970    \let\bbl@s@activate\bbl@activate
971    \def\bbl@activate#1{%
972      \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
973    \let\bbl@s@deactivate\bbl@deactivate
974    \def\bbl@deactivate#1{%
975      \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
976 \fi
```

\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.

```
977 \def\bbl@prim@s{%
978    \prime\futurelet\@let@token\bbl@pr@m@s}
979 \def\bbl@if@primes#1#2{%
980    \ifx#1\@let@token
981      \expandafter\@firstoftwo
982    \else\ifx#2\@let@token
983      \bbl@afterelse\expandafter\@firstoftwo
984    \else
985      \bbl@afterfi\expandafter\@secondoftwo
986    \fi\fi}
987 \begingroup
988    \catcode`\^=7  \catcode`\*=\active  \lccode`\*=`\^
989    \catcode`\'=12 \catcode`\"=\active  \lccode`\"=`\'
990    \lowercase{%
991      \gdef\bbl@pr@m@s{%
992        \bbl@if@primes"'%
993          \pr@@@s
994          {\bbl@if@primes*^\pr@@@t\egroup}}}
995 \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).

```
996 \initiate@active@char{~}
997 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
998 \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.

```
999  \expandafter\def\csname OT1dqpos\endcsname{127}
1000 \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

```
1001 \ifx\f@encoding\@undefined
1002   \def\f@encoding{OT1}
1003 \fi
```

### 8.5 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.

```
1004 \bbl@trace{Language attributes}
1005 \newcommand\languageattribute[2]{%
1006   \def\bbl@tempc{#1}%
1007   \bbl@fixname\bbl@tempc
1008   \bbl@iflanguage\bbl@tempc{%
1009     \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.

```
1010       \ifx\bbl@known@attribs\@undefined
1011         \in@false
1012       \else
```

Now we need to see if the attribute occurs in the list of already selected attributes.

```
1013         \bbl@xin@{,\bbl@tempc-##1,}{,\bbl@known@attribs,}%
1014       \fi
```

When the attribute was in the list we issue a warning; this might not be the users intention.

```
1015       \ifin@
1016         \bbl@warning{%
1017           You have more than once selected the attribute '##1'\\%
1018           for language #1}%
1019       \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.

```
1020         \bbl@exp{%
1021           \\\bbl@add@list\\\bbl@known@attribs{\bbl@tempc-##1}}%
1022         \edef\bbl@tempa{\bbl@tempc-##1}%
1023         \expandafter\bbl@ifknown@ttrib\expandafter{\bbl@tempa}\bbl@attributes%
1024         {\csname\bbl@tempc @attr@##1\endcsname}%
1025         {\@attrerr{\bbl@tempc}{##1}}%
1026     \fi}}}
```

This command should only be used in the preamble of a document.

```
1027 \@onlypreamble\languageattribute
```

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

```
1028 \newcommand*{\@attrerr}[2]{%
1029   \bbl@error
1030     {The attribute #2 is unknown for language #1.}%
1031     {Your command will be ignored, type <return> to proceed}}
```

<dl>
<dt>\bbl@declare@ttribute</dt>
<dd>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}.</dd>
</dl>

```
1032 \def\bbl@declare@ttribute#1#2#3{%
1033   \bbl@xin@{,#2,}{,\BabelModifiers,}%
1034   \ifin@
1035     \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
1036   \fi
1037   \bbl@add@list\bbl@attributes{#1-#2}%
1038   \expandafter\def\csname#1@attr@#2\endcsname{#3}}
```

<dl>
<dt>\bbl@ifattributeset</dt>
<dd>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.</dd>
</dl>

```
1039 \def\bbl@ifattributeset#1#2#3#4{%
```

First we need to find out if any attributes were set; if not we're done.

```
1040   \ifx\bbl@known@attribs\@undefined
1041     \in@false
1042   \else
```

The we need to check the list of known attributes.

```
1043     \bbl@xin@{,#1-#2,}{,\bbl@known@attribs,}%
1044   \fi
```

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'.

```
1045   \ifin@
1046     \bbl@afterelse#3%
1047   \else
1048     \bbl@afterfi#4%
1049   \fi
1050   }
```

<dl>
<dt>\bbl@ifknown@ttrib</dt>
<dd>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.</dd>
</dl>

```
1051 \def\bbl@ifknown@ttrib#1#2{%
```

We first assume the attribute is unknown.

```
1052   \let\bbl@tempa\@secondoftwo
```

Then we loop over the list of known attributes, trying to find a match.

```
1053   \bbl@loopx\bbl@tempb{#2}{%
1054     \expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{,#1,}%
1055     \ifin@
```

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

```
1056       \let\bbl@tempa\@firstoftwo
1057     \else
1058     \fi}%
```

Finally we execute \bbl@tempa.

```
1059    \bbl@tempa
1060 }
```

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

```
1061 \def\bbl@clear@ttribs{%
1062    \ifx\bbl@attributes\@undefined\else
1063       \bbl@loopx\bbl@tempa{\bbl@attributes}{%
1064          \expandafter\bbl@clear@ttrib\bbl@tempa.
1065          }%
1066       \let\bbl@attributes\@undefined
1067    \fi}
1068 \def\bbl@clear@ttrib#1-#2.{%
1069    \expandafter\let\csname#1@attr@#2\endcsname\@undefined}
1070 \AtBeginDocument{\bbl@clear@ttribs}
```

## 8.6   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
```
1071 \bbl@trace{Macros for saving definitions}
1072 \def\babel@beginsave{\babel@savecnt\z@}
```

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

```
1073 \newcount\babel@savecnt
1074 \babel@beginsave
```

\babel@save    The macro \babel@save⟨csname⟩ saves the current meaning of the control sequence ⟨csname⟩ to \originalTeX[30]. 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.

```
1075 \def\babel@save#1{%
1076    \expandafter\let\csname babel@\number\babel@savecnt\endcsname#1\relax
1077    \toks@\expandafter{\originalTeX\let#1=}%
1078    \bbl@exp{%
1079       \def\\\originalTeX{\the\toks@\<babel@\number\babel@savecnt>\relax}}%
1080    \advance\babel@savecnt\@ne}
```

\babel@savevariable    The macro \babel@savevariable⟨variable⟩ saves the value of the variable. ⟨variable⟩ can be anything allowed after the \the primitive.

```
1081 \def\babel@savevariable#1{%
1082    \toks@\expandafter{\originalTeX #1=}%
1083    \bbl@exp{\def\\\originalTeX{\the\toks@\the#1\relax}}}
```

---

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

78

\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.

```
1084 \def\bbl@frenchspacing{%
1085   \ifnum\the\sfcode`\.=\@m
1086     \let\bbl@nonfrenchspacing\relax
1087   \else
1088     \frenchspacing
1089     \let\bbl@nonfrenchspacing\nonfrenchspacing
1090   \fi}
1091 \let\bbl@nonfrenchspacing\nonfrenchspacing
```

## 8.7 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.

```
1092 \bbl@trace{Short tags}
1093 \def\babeltags#1{%
1094   \edef\bbl@tempa{\zap@space#1 \@empty}%
1095   \def\bbl@tempb##1=##2\@@{%
1096     \edef\bbl@tempc{
1097       \noexpand\newcommand
1098       \expandafter\noexpand\csname ##1\endcsname{%
1099         \noexpand\protect
1100         \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}
1101       \noexpand\newcommand
1102       \expandafter\noexpand\csname text##1\endcsname{%
1103         \noexpand\foreignlanguage{##2}}}%
1104     \bbl@tempc}%
1105   \bbl@for\bbl@tempa\bbl@tempa{%
1106     \expandafter\bbl@tempb\bbl@tempa\@@}}
```

## 8.8 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.

```
1107 \bbl@trace{Hyphens}
1108 \@onlypreamble\babelhyphenation
1109 \AtEndOfPackage{%
1110   \newcommand\babelhyphenation[2][\@empty]{%
1111     \ifx\bbl@hyphenation@\relax
1112       \let\bbl@hyphenation@\@empty
1113     \fi
1114     \ifx\bbl@hyphlist\@empty\else
1115       \bbl@warning{%
1116         You must not intermingle \string\selectlanguage\space and\\%
1117         \string\babelhyphenation\space or some exceptions will not\\%
1118         be taken into account. Reported}%
1119     \fi
1120     \ifx\@empty#1%
1121       \protected@edef\bbl@hyphenation@{\bbl@hyphenation@\space#2}%
1122     \else
1123       \bbl@vforeach{#1}{%
```

```
1124        \def\bbl@tempa{##1}%
1125        \bbl@fixname\bbl@tempa
1126        \bbl@iflanguage\bbl@tempa{%
1127          \bbl@csarg\protected@edef{hyphenation@\bbl@tempa}{%
1128            \bbl@ifunset{bbl@hyphenation@\bbl@tempa}%
1129              \@empty
1130              {\csname bbl@hyphenation@\bbl@tempa\endcsname\space}%
1131            #2}}}%
1132      \fi}}
```

\bbl@allowhyphens This macro makes hyphenation possible. Basically its definition is nothing more than
\nobreak \hskip 0pt plus 0pt[31].

```
1133 \def\bbl@allowhyphens{\ifvmode\else\nobreak\hskip\z@skip\fi}
1134 \def\bbl@t@one{T1}
1135 \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.

```
1136 \newcommand\babelnullhyphen{\char\hyphenchar\font}
1137 \def\babelhyphen{\active@prefix\babelhyphen\bbl@hyphen}
1138 \def\bbl@hyphen{%
1139   \@ifstar{\bbl@hyphen@i @}{\bbl@hyphen@i\@empty}}
1140 \def\bbl@hyphen@i#1#2{%
1141   \bbl@ifunset{bbl@hy@#1#2\@empty}%
1142     {\csname bbl@#1usehyphen\endcsname{\discretionary{#2}{}{#2}}}%
1143     {\csname bbl@hy@#1#2\@empty\endcsname}}
```

The following two commands are used to wrap the "hyphen" and set the behavior of the
rest of the word – the version with a single @ is used when further hyphenation is allowed,
while that with @@ if no more hyphen 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 discretionaty 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.

```
1144 \def\bbl@usehyphen#1{%
1145   \leavevmode
1146   \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
1147   \nobreak\hskip\z@skip}
1148 \def\bbl@@usehyphen#1{%
1149   \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}
```

The following macro inserts the hyphen char.

```
1150 \def\bbl@hyphenchar{%
1151   \ifnum\hyphenchar\font=\m@ne
1152     \babelnullhyphen
1153   \else
1154     \char\hyphenchar\font
1155   \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.

```
1156 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}{}}}
1157 \def\bbl@hy@@soft{\bbl@@usehyphen{\discretionary{\bbl@hyphenchar}{}{}}}
1158 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
1159 \def\bbl@hy@@hard{\bbl@@usehyphen\bbl@hyphenchar}
```

---

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

```
1160 \def\bbl@hy@nobreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}}
1161 \def\bbl@hy@@nobreak{\mbox{\bbl@hyphenchar}}
1162 \def\bbl@hy@repeat{%
1163   \bbl@usehyphen{%
1164     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
1165 \def\bbl@hy@@repeat{%
1166   \bbl@@usehyphen{%
1167     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
1168 \def\bbl@hy@empty{\hskip\z@skip}
1169 \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.

```
1170 \def\bbl@disc#1#2{\nobreak\discretionary{#2-}{}{#1}\bbl@allowhyphens}
```

## 8.9   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 ued 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.

```
1171 \bbl@trace{Multiencoding strings}
1172 \def\bbl@toglobal#1{\global\let#1#1}
1173 \def\bbl@recatcode#1{%
1174   \@tempcnta="7F
1175   \def\bbl@tempa{%
1176     \ifnum\@tempcnta>"FF\else
1177       \catcode\@tempcnta=#1\relax
1178       \advance\@tempcnta\@ne
1179       \expandafter\bbl@tempa
1180     \fi}%
1181   \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).

```
1182 \@ifpackagewith{babel}{nocase}%
1183   {\let\bbl@patchuclc\relax}%
1184   {\def\bbl@patchuclc{%
1185     \global\let\bbl@patchuclc\relax
1186     \g@addto@macro\@uclclist{\reserved@b{\reserved@b\bbl@uclc}}%
1187     \gdef\bbl@uclc##1{%
1188       \let\bbl@encoded\bbl@encoded@uclc
1189       \bbl@ifunset{\languagename @bbl@uclc}% and resumes it
1190         {##1}%
```

```
1191        {\let\bbl@tempa##1\relax % Used by LANG@bbl@uclc
1192         \csname\languagename @bbl@uclc\endcsname}%
1193       {\bbl@tolower\@empty}{\bbl@toupper\@empty}}%
1194     \gdef\bbl@tolower{\csname\languagename @bbl@lc\endcsname}%
1195     \gdef\bbl@toupper{\csname\languagename @bbl@uc\endcsname}}}
```

1196 ⟨⟨*More package options⟩⟩ ≡
```
1197 \DeclareOption{nocase}{}
```
1198 ⟨⟨/More package options⟩⟩

The following package options control the behavior of \SetString.

1199 ⟨⟨*More package options⟩⟩ ≡
```
1200 \let\bbl@opt@strings\@nnil % accept strings=value
1201 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
1202 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
1203 \def\BabelStringsDefault{generic}
```
1204 ⟨⟨/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.

```
1205 \@onlypreamble\StartBabelCommands
1206 \def\StartBabelCommands{%
1207   \begingroup
1208   \bbl@recatcode{11}%
1209   ⟨⟨Macros local to BabelCommands⟩⟩
1210   \def\bbl@provstring##1##2{%
1211     \providecommand##1{##2}%
1212     \bbl@toglobal##1}%
1213   \global\let\bbl@scafter\@empty
1214   \let\StartBabelCommands\bbl@startcmds
1215   \ifx\BabelLanguages\relax
1216     \let\BabelLanguages\CurrentOption
1217   \fi
1218   \begingroup
1219   \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
1220   \StartBabelCommands}
1221 \def\bbl@startcmds{%
1222   \ifx\bbl@screset\@nnil\else
1223     \bbl@usehooks{stopcommands}{}%
1224   \fi
1225   \endgroup
1226   \begingroup
1227   \@ifstar
1228     {\ifx\bbl@opt@strings\@nnil
1229        \let\bbl@opt@strings\BabelStringsDefault
1230      \fi
1231      \bbl@startcmds@i}%
1232     \bbl@startcmds@i}
1233 \def\bbl@startcmds@i#1#2{%
1234   \edef\bbl@L{\zap@space#1 \@empty}%
1235   \edef\bbl@G{\zap@space#2 \@empty}%
1236   \bbl@startcmds@ii}
```

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.

```
1237 \newcommand\bbl@startcmds@ii[1][\@empty]{%
1238   \let\SetString\@gobbletwo
1239   \let\bbl@stringdef\@gobbletwo
1240   \let\AfterBabelCommands\@gobble
1241   \ifx\@empty#1%
1242     \def\bbl@sc@label{generic}%
1243     \def\bbl@encstring##1##2{%
1244       \ProvideTextCommandDefault##1{##2}%
1245       \bbl@toglobal##1%
1246       \expandafter\bbl@toglobal\csname\string?\string##1\endcsname}%
1247     \let\bbl@sctest\in@true
1248   \else
1249     \let\bbl@sc@charset\space % <- zapped below
1250     \let\bbl@sc@fontenc\space % <-   "        "
1251     \def\bbl@tempa##1=##2\@nil{%
1252       \bbl@csarg\edef{sc@\zap@space##1 \@empty}{##2 }}%
1253     \bbl@vforeach{label=#1}{\bbl@tempa##1\@nil}%
1254     \def\bbl@tempa##1 ##2{% space -> comma
1255       ##1%
1256       \ifx\@empty##2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa##2\fi}%
1257     \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%
1258     \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%
1259     \edef\bbl@sc@charset{\expandafter\zap@space\bbl@sc@charset\@empty}%
1260     \def\bbl@encstring##1##2{%
1261       \bbl@foreach\bbl@sc@fontenc{%
1262         \bbl@ifunset{T@####1}%
1263           {}%
1264           {\ProvideTextCommand##1{####1}{##2}%
1265            \bbl@toglobal##1%
1266            \expandafter
1267            \bbl@toglobal\csname####1\string##1\endcsname}}%
1268     \def\bbl@sctest{%
1269       \bbl@xin@{,\bbl@opt@strings,}{,\bbl@sc@label,\bbl@sc@fontenc,}}%
1270   \fi
1271   \ifx\bbl@opt@strings\@nnil        % ie, no strings key -> defaults
1272   \else\ifx\bbl@opt@strings\relax   % ie, strings=encoded
1273     \let\AfterBabelCommands\bbl@aftercmds
1274     \let\SetString\bbl@setstring
1275     \let\bbl@stringdef\bbl@encstring
1276   \else      % ie, strings=value
1277   \bbl@sctest
1278   \ifin@
1279     \let\AfterBabelCommands\bbl@aftercmds
1280     \let\SetString\bbl@setstring
1281     \let\bbl@stringdef\bbl@provstring
1282   \fi\fi\fi
1283   \bbl@scswitch
1284   \ifx\bbl@G\@empty
1285     \def\SetString##1##2{%
1286       \bbl@error{Missing group for string \string##1}%
1287         {You must assign strings to some category, typically\\%
1288          captions or extras, but you set none}}%
1289   \fi
```

```
1290    \ifx\@empty#1%
1291      \bbl@usehooks{defaultcommands}{}%
1292    \else
1293      \@expandtwoargs
1294      \bbl@usehooks{encodedcommands}{{\bbl@sc@charset}{\bbl@sc@fontenc}}%
1295    \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) .

```
1296 \def\bbl@forlang#1#2{%
1297    \bbl@for#1\bbl@L{%
1298      \bbl@xin@{,#1,}{,\BabelLanguages,}%
1299      \ifin@#2\relax\fi}}
1300 \def\bbl@scswitch{%
1301    \bbl@forlang\bbl@tempa{%
1302      \ifx\bbl@G\@empty\else
1303        \ifx\SetString\@gobbletwo\else
1304          \edef\bbl@GL{\bbl@G\bbl@tempa}%
1305          \bbl@xin@{,\bbl@GL,}{,\bbl@screset,}%
1306          \ifin@\else
1307            \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
1308            \xdef\bbl@screset{\bbl@screset,\bbl@GL}%
1309          \fi
1310        \fi
1311      \fi}}
1312 \AtEndOfPackage{%
1313    \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{}{#2}}}%
1314    \let\bbl@scswitch\relax}
1315 \@onlypreamble\EndBabelCommands
1316 \def\EndBabelCommands{%
1317    \bbl@usehooks{stopcommands}{}%
1318    \endgroup
1319    \endgroup
1320    \bbl@scafter}
```

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.

```
1321 \def\bbl@setstring#1#2{%
1322    \bbl@forlang\bbl@tempa{%
1323      \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
1324      \bbl@ifunset{\bbl@LC}% eg, \germanchaptername
1325        {\global\expandafter  % TODO - con \bbl@exp ?
1326          \bbl@add\csname\bbl@G\bbl@tempa\expandafter\endcsname\expandafter
1327            {\expandafter\bbl@scset\expandafter#1\csname\bbl@LC\endcsname}}%
1328        {}%
1329      \def\BabelString{#2}%
```

```
1330      \bbl@usehooks{stringprocess}{}%
1331      \expandafter\bbl@stringdef
1332        \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.

```
1333 \ifx\bbl@opt@strings\relax
1334   \def\bbl@scset#1#2{\def#1{\bbl@encoded#2}}
1335   \bbl@patchuclc
1336   \let\bbl@encoded\relax
1337   \def\bbl@encoded@uclc#1{%
1338     \@inmathwarn#1%
1339     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
1340       \expandafter\ifx\csname ?\string#1\endcsname\relax
1341         \TextSymbolUnavailable#1%
1342       \else
1343         \csname ?\string#1\endcsname
1344       \fi
1345     \else
1346       \csname\cf@encoding\string#1\endcsname
1347     \fi}
1348 \else
1349   \def\bbl@scset#1#2{\def#1{#2}}
1350 \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.

```
1351 ⟨⟨∗Macros local to BabelCommands⟩⟩ ≡
1352 \def\SetStringLoop##1##2{%
1353     \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
1354     \count@\z@
1355     \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
1356       \advance\count@\@ne
1357       \toks@\expandafter{\bbl@tempa}%
1358       \bbl@exp{%
1359         \\\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
1360         \count@=\the\count@\relax}}}%
1361 ⟨⟨/Macros local to BabelCommands⟩⟩
```

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

```
1362 \def\bbl@aftercmds#1{%
1363   \toks@\expandafter{\bbl@scafter#1}%
1364   \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.

```
1365 ⟨⟨∗Macros local to BabelCommands⟩⟩ ≡
1366   \newcommand\SetCase[3][]{%
1367     \bbl@patchuclc
1368     \bbl@forlang\bbl@tempa{%
1369       \expandafter\bbl@encstring
```

```
1370        \csname\bbl@tempa @bbl@uclc\endcsname{\bbl@tempa##1}%
1371      \expandafter\bbl@encstring
1372        \csname\bbl@tempa @bbl@uc\endcsname{##2}%
1373      \expandafter\bbl@encstring
1374        \csname\bbl@tempa @bbl@lc\endcsname{##3}}}%
1375 ⟨⟨/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.

```
1376 ⟨⟨∗Macros local to BabelCommands⟩⟩ ≡
1377    \newcommand\SetHyphenMap[1]{%
1378      \bbl@forlang\bbl@tempa{%
1379        \expandafter\bbl@stringdef
1380          \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}
1381 ⟨⟨/Macros local to BabelCommands⟩⟩
```

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

```
1382 \newcommand\BabelLower[2]{% one to one.
1383    \ifnum\lccode#1=#2\else
1384      \babel@savevariable{\lccode#1}%
1385      \lccode#1=#2\relax
1386    \fi}
1387 \newcommand\BabelLowerMM[4]{% many-to-many
1388    \@tempcnta=#1\relax
1389    \@tempcntb=#4\relax
1390    \def\bbl@tempa{%
1391      \ifnum\@tempcnta>#2\else
1392        \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
1393        \advance\@tempcnta#3\relax
1394        \advance\@tempcntb#3\relax
1395        \expandafter\bbl@tempa
1396      \fi}%
1397    \bbl@tempa}
1398 \newcommand\BabelLowerMO[4]{% many-to-one
1399    \@tempcnta=#1\relax
1400    \def\bbl@tempa{%
1401      \ifnum\@tempcnta>#2\else
1402        \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
1403        \advance\@tempcnta#3
1404        \expandafter\bbl@tempa
1405      \fi}%
1406    \bbl@tempa}
```

The following package options control the behavior of hyphenation mapping.

```
1407 ⟨⟨∗More package options⟩⟩ ≡
1408 \DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\z@}
1409 \DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap\@ne}
1410 \DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\tw@}
1411 \DeclareOption{hyphenmap=other}{\chardef\bbl@opt@hyphenmap\thr@@}
1412 \DeclareOption{hyphenmap=other*}{\chardef\bbl@opt@hyphenmap4\relax}
1413 ⟨⟨/More package options⟩⟩
```

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

```
1414 \AtEndOfPackage{%
1415    \ifx\bbl@opt@hyphenmap\@undefined
1416      \bbl@xin@{,}{\bbl@language@opts}%
1417      \chardef\bbl@opt@hyphenmap\ifin@4\else\@ne\fi
1418    \fi}
```

## 8.10  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.

```
1419 \bbl@trace{Macros related to glyphs}
1420 \def\set@low@box#1{\setbox\tw@\hbox{,}\setbox\z@\hbox{#1}%
1421     \dimen\z@\ht\z@ \advance\dimen\z@ -\ht\tw@%
1422     \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.

```
1423 \def\save@sf@q#1{\leavevmode
1424     \begingroup
1425         \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
1426     \endgroup}
```

## 8.11  Making glyphs available

This section makes a number of glyphs available that either do not exist in the OT1 encoding and have to be 'faked', or that are not accessible through T1enc.def.

### 8.11.1  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.

```
1427 \ProvideTextCommand{\quotedblbase}{OT1}{%
1428     \save@sf@q{\set@low@box{\textquotedblright\/}%
1429         \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.

```
1430 \ProvideTextCommandDefault{\quotedblbase}{%
1431     \UseTextSymbol{OT1}{\quotedblbase}}
```

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

```
1432 \ProvideTextCommand{\quotesinglbase}{OT1}{%
1433     \save@sf@q{\set@low@box{\textquoteright\/}%
1434         \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.

```
1435 \ProvideTextCommandDefault{\quotesinglbase}{%
1436     \UseTextSymbol{OT1}{\quotesinglbase}}
```

\guillemotleft    The guillemet characters are not available in OT1 encoding. They are faked.
\guillemotright
```
1437 \ProvideTextCommand{\guillemotleft}{OT1}{%
1438     \ifmmode
1439         \ll
1440     \else
1441         \save@sf@q{\nobreak
1442             \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
1443     \fi}
1444 \ProvideTextCommand{\guillemotright}{OT1}{%
1445     \ifmmode
1446         \gg
1447     \else
```

```
1448        \save@sf@q{\nobreak
1449          \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
1450    \fi}
```

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

```
1451 \ProvideTextCommandDefault{\guillemotleft}{%
1452    \UseTextSymbol{OT1}{\guillemotleft}}
1453 \ProvideTextCommandDefault{\guillemotright}{%
1454    \UseTextSymbol{OT1}{\guillemotright}}
```

\guilsinglleft   The single guillemets are not available in OT1 encoding. They are faked.
\guilsinglright
```
1455 \ProvideTextCommand{\guilsinglleft}{OT1}{%
1456    \ifmmode
1457      <%
1458    \else
1459      \save@sf@q{\nobreak
1460        \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
1461    \fi}
1462 \ProvideTextCommand{\guilsinglright}{OT1}{%
1463    \ifmmode
1464      >%
1465    \else
1466      \save@sf@q{\nobreak
1467        \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
1468    \fi}
```

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

```
1469 \ProvideTextCommandDefault{\guilsinglleft}{%
1470    \UseTextSymbol{OT1}{\guilsinglleft}}
1471 \ProvideTextCommandDefault{\guilsinglright}{%
1472    \UseTextSymbol{OT1}{\guilsinglright}}
```

### 8.11.2   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.

```
1473 \DeclareTextCommand{\ij}{OT1}{%
1474    i\kern-0.02em\bbl@allowhyphens j}
1475 \DeclareTextCommand{\IJ}{OT1}{%
1476    I\kern-0.02em\bbl@allowhyphens J}
1477 \DeclareTextCommand{\ij}{T1}{\char188}
1478 \DeclareTextCommand{\IJ}{T1}{\char156}
```

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

```
1479 \ProvideTextCommandDefault{\ij}{%
1480    \UseTextSymbol{OT1}{\ij}}
1481 \ProvideTextCommandDefault{\IJ}{%
1482    \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
     Stipcevic Mario, (stipcevic@olimp.irb.hr).

```
1483 \def\crrtic@{\hrule height0.1ex width0.3em}
```

```
1484 \def\crttic@{\hrule height0.1ex width0.33em}
1485 \def\ddj@{%
1486   \setbox0\hbox{d}\dimen@=\ht0
1487   \advance\dimen@1ex
1488   \dimen@.45\dimen@
1489   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
1490   \advance\dimen@ii.5ex
1491   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
1492 \def\DDJ@{%
1493   \setbox0\hbox{D}\dimen@=.55\ht0
1494   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
1495   \advance\dimen@ii.15ex %              correction for the dash position
1496   \advance\dimen@ii-.15\fontdimen7\font %    correction for cmtt font
1497   \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
1498   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
1499 %
1500 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
1501 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}
```

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

```
1502 \ProvideTextCommandDefault{\dj}{%
1503   \UseTextSymbol{OT1}{\dj}}
1504 \ProvideTextCommandDefault{\DJ}{%
1505   \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.

```
1506 \DeclareTextCommand{\SS}{OT1}{SS}
1507 \ProvideTextCommandDefault{\SS}{\UseTextSymbol{OT1}{\SS}}
```

### 8.11.3  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
```
1508 \ProvideTextCommandDefault{\glq}{%
1509   \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}
```

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

```
1510 \ProvideTextCommand{\grq}{T1}{%
1511   \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
1512 \ProvideTextCommand{\grq}{TU}{%
1513   \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
1514 \ProvideTextCommand{\grq}{OT1}{%
1515   \save@sf@q{\kern-.0125em
1516     \textormath{\textquoteleft}{\mbox{\textquoteleft}}%
1517     \kern.07em\relax}}
1518 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}\grq}
```

\glqq  The 'german' double quotes.

\grqq
```
1519 \ProvideTextCommandDefault{\glqq}{%
1520   \textormath{\quotedblbase}{\mbox{\quotedblbase}}}
```

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

```
1521 \ProvideTextCommand{\grqq}{T1}{%
1522   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
1523 \ProvideTextCommand{\grqq}{TU}{%
1524   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
1525 \ProvideTextCommand{\grqq}{OT1}{%
1526   \save@sf@q{\kern-.07em
1527     \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}%
1528     \kern.07em\relax}}
1529 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}
```

`\flq`  The 'french' single guillemets.
`\frq`
```
1530 \ProvideTextCommandDefault{\flq}{%
1531   \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
1532 \ProvideTextCommandDefault{\frq}{%
1533   \textormath{\guilsinglright}{\mbox{\guilsinglright}}}
```

`\flqq`  The 'french' double guillemets.
`\frqq`
```
1534 \ProvideTextCommandDefault{\flqq}{%
1535   \textormath{\guillemotleft}{\mbox{\guillemotleft}}}
1536 \ProvideTextCommandDefault{\frqq}{%
1537   \textormath{\guillemotright}{\mbox{\guillemotright}}}
```

### 8.11.4  Umlauts and tremas

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

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

```
1538 \def\umlauthigh{%
1539   \def\bbl@umlauta##1{\leavevmode\bgroup%
1540     \expandafter\accent\csname\f@encoding dqpos\endcsname
1541     ##1\bbl@allowhyphens\egroup}%
1542   \let\bbl@umlaute\bbl@umlauta}
1543 \def\umlautlow{%
1544   \def\bbl@umlauta{\protect\lower@umlaut}}
1545 \def\umlautelow{%
1546   \def\bbl@umlaute{\protect\lower@umlaut}}
1547 \umlauthigh
```

`\lower@umlaut`  The command `\lower@umlaut` is used to position the `\"` closer to the letter.
We want the umlaut character lowered, nearer to the letter. To do this we need an extra ⟨*dimen*⟩ register.

```
1548 \expandafter\ifx\csname U@D\endcsname\relax
1549   \csname newdimen\endcsname\U@D
1550 \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.

```
1551 \def\lower@umlaut#1{%
1552   \leavevmode\bgroup
1553     \U@D 1ex%
1554     {\setbox\z@\hbox{%
1555       \expandafter\char\csname\f@encoding dqpos\endcsname}%
1556       \dimen@ -.45ex\advance\dimen@\ht\z@
1557       \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}%
1558     \expandafter\accent\csname\f@encoding dqpos\endcsname
1559     \fontdimen5\font\U@D #1%
1560   \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).

```
1561 \AtBeginDocument{%
1562   \DeclareTextCompositeCommand{\"}{OT1}{a}{\bbl@umlauta{a}}%
1563   \DeclareTextCompositeCommand{\"}{OT1}{e}{\bbl@umlaute{e}}%
1564   \DeclareTextCompositeCommand{\"}{OT1}{i}{\bbl@umlaute{\i}}%
1565   \DeclareTextCompositeCommand{\"}{OT1}{\i}{\bbl@umlaute{\i}}%
1566   \DeclareTextCompositeCommand{\"}{OT1}{o}{\bbl@umlauta{o}}%
1567   \DeclareTextCompositeCommand{\"}{OT1}{u}{\bbl@umlauta{u}}%
1568   \DeclareTextCompositeCommand{\"}{OT1}{A}{\bbl@umlauta{A}}%
1569   \DeclareTextCompositeCommand{\"}{OT1}{E}{\bbl@umlaute{E}}%
1570   \DeclareTextCompositeCommand{\"}{OT1}{I}{\bbl@umlaute{I}}%
1571   \DeclareTextCompositeCommand{\"}{OT1}{O}{\bbl@umlauta{O}}%
1572   \DeclareTextCompositeCommand{\"}{OT1}{U}{\bbl@umlauta{U}}%
1573 }
```

Finally, the default is to use English as the main language.

```
1574 \ifx\l@english\@undefined
1575   \chardef\l@english\z@
1576 \fi
1577 \main@language{english}
```

## 8.12   Layout

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

```
1578 \bbl@trace{Bidi layout}
1579 \providecommand\IfBabelLayout[3]{#3}%
1580 \newcommand\BabelPatchSection[1]{%
1581   \@ifundefined{#1}{}{%
1582     \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
1583     \@namedef{#1}{%
1584       \@ifstar{\bbl@presec@s{#1}}%
1585               {\@dblarg{\bbl@presec@x{#1}}}}}}
1586 \def\bbl@presec@x#1[#2]#3{%
1587   \bbl@exp{%
```

```
1588      \\\select@language@x{\bbl@main@language}%
1589      \\\@nameuse{bbl@sspre@#1}%
1590      \\\@nameuse{bbl@ss@#1}%
1591        [\\\foreignlanguage{\languagename}{\unexpanded{#2}}]%
1592        {\\\foreignlanguage{\languagename}{\unexpanded{#3}}}%
1593      \\\select@language@x{\languagename}}}
1594 \def\bbl@presec@s#1#2{%
1595   \bbl@exp{%
1596      \\\select@language@x{\bbl@main@language}%
1597      \\\@nameuse{bbl@sspre@#1}%
1598      \\\@nameuse{bbl@ss@#1}*%
1599        {\\\foreignlanguage{\languagename}{\unexpanded{#2}}}%
1600      \\\select@language@x{\languagename}}}
1601 \IfBabelLayout{sectioning}%
1602   {\BabelPatchSection{part}%
1603    \BabelPatchSection{chapter}%
1604    \BabelPatchSection{section}%
1605    \BabelPatchSection{subsection}%
1606    \BabelPatchSection{subsubsection}%
1607    \BabelPatchSection{paragraph}%
1608    \BabelPatchSection{subparagraph}%
1609    \def\babel@toc#1{%
1610      \select@language@x{\bbl@main@language}}}{}
1611 \IfBabelLayout{captions}%
1612   {\BabelPatchSection{caption}}{}
```

Now we load definition files for engines.

```
1613 \bbl@trace{Input engine specific macros}
1614 \ifcase\bbl@engine
1615   \input txtbabel.def
1616 \or
1617   \input luababel.def
1618 \or
1619   \input xebabel.def
1620 \fi
```

### 8.13   Creating languages

\babelprovide is a general purpose tool for creating languages. Currently it just creates
the language infrastructure, but in the future it will be able to read data from ini files, as
well as to create variants. Unlike the nil pseudo-language, captions are defined, but with a
warning to invite the user to provide the real string.

```
1621 \bbl@trace{Creating languages and reading ini files}
1622 \newcommand\babelprovide[2][]{%
1623   \let\bbl@savelangname\languagename
1624   \def\languagename{#2}%
1625   \let\bbl@KVP@captions\@nil
1626   \let\bbl@KVP@import\@nil
1627   \let\bbl@KVP@main\@nil
1628   \let\bbl@KVP@script\@nil
1629   \let\bbl@KVP@language\@nil
1630   \let\bbl@KVP@dir\@nil
1631   \let\bbl@KVP@hyphenrules\@nil
1632   \let\bbl@KVP@mapfont\@nil
1633   \bbl@forkv{#1}{\bbl@csarg\def{KVP@##1}{##2}}%  TODO - error handling
1634   \ifx\bbl@KVP@captions\@nil
1635     \let\bbl@KVP@captions\bbl@KVP@import
1636   \fi
```

```
1637    \bbl@ifunset{date#2}%
1638      {\bbl@provide@new{#2}}%
1639      {\bbl@ifblank{#1}%
1640        {\bbl@error
1641          {If you want to modify `#2' you must tell how in\\%
1642           the optional argument. Currently there are three\\%
1643           options: captions=lang-tag, hyphenrules=lang-list\\%
1644           import=lang-tag}%
1645          {Use this macro as documented}}%
1646        {\bbl@provide@renew{#2}}}%
1647    \bbl@exp{\\\babelensure[exclude=\\\today]{#2}}%
1648    \bbl@ifunset{bbl@ensure@\languagename}%
1649      {\bbl@exp{%
1650        \\\DeclareRobustCommand\<bbl@ensure@\languagename>[1]{%
1651          \\\foreignlanguage{\languagename}%
1652          {####1}}}}%
1653      {}%
1654    \ifx\bbl@KVP@script\@nil\else
1655      \bbl@csarg\edef{sname@#2}{\bbl@KVP@script}%
1656    \fi
1657    \ifx\bbl@KVP@language\@nil\else
1658      \bbl@csarg\edef{lname@#2}{\bbl@KVP@language}%
1659    \fi
1660    \ifx\bbl@KVP@mapfont\@nil\else
1661      \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}{}%
1662        {\bbl@error{Option `\bbl@KVP@mapfont' unknown for\\%
1663                    mapfont. Use `direction'.%
1664                    {See the manual for details.}}}%
1665      \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
1666      \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
1667      \ifx\bbl@mapselect\@undefined
1668        \AtBeginDocument{%
1669          \expandafter\bbl@add\csname selectfont \endcsname{{\bbl@mapselect}}%
1670          {\selectfont}}%
1671        \def\bbl@mapselect{%
1672          \let\bbl@mapselect\relax
1673          \edef\bbl@prefontid{\fontid\font}}%
1674        \def\bbl@mapdir##1{%
1675          {\def\languagename{##1}\bbl@switchfont
1676           \directlua{Babel.fontmap
1677             [\the\csname bbl@wdir@##1\endcsname]%
1678             [\bbl@prefontid]=\fontid\font}}}%
1679      \fi
1680      \bbl@exp{\\\bbl@add\\\bbl@mapselect{\\\bbl@mapdir{\languagename}}}%
1681    \fi
1682    \let\languagename\bbl@savelangname}
```

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

```
1683 \def\bbl@provide@new#1{%
1684   \@namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
1685   \@namedef{extras#1}{}%
1686   \@namedef{noextras#1}{}%
1687   \StartBabelCommands*{#1}{captions}%
1688     \ifx\bbl@KVP@captions\@nil %      and also if import, implicit
1689       \def\bbl@tempb##1{%              elt for \bbl@captionslist
1690         \ifx##1\@empty\else
1691           \bbl@exp{%
1692             \\\SetString\\##1{%
1693               \\\bbl@nocaption{\bbl@stripslash##1}{\<#1\bbl@stripslash##1>}}}%
```

93

```
1694        \expandafter\bbl@tempb
1695      \fi}%
1696    \expandafter\bbl@tempb\bbl@captionslist\@empty
1697  \else
1698    \bbl@read@ini{\bbl@KVP@captions}%  Here all letters cat = 11
1699    \bbl@after@ini
1700    \bbl@savestrings
1701  \fi
1702  \StartBabelCommands*{#1}{date}%
1703    \ifx\bbl@KVP@import\@nil
1704      \bbl@exp{%
1705        \\\SetString\\\today{\\\bbl@nocaption{today}{\<#1today>}}}%
1706    \else
1707      \bbl@savetoday
1708      \bbl@savedate
1709    \fi
1710  \EndBabelCommands
1711  \bbl@exp{%
1712    \def\<#1hyphenmins>{%
1713      {\bbl@ifunset{bbl@lfthm@#1}{2}{\@nameuse{bbl@lfthm@#1}}}%
1714      {\bbl@ifunset{bbl@rgthm@#1}{3}{\@nameuse{bbl@rgthm@#1}}}}}%
1715  \bbl@provide@hyphens{#1}%
1716  \ifx\bbl@KVP@main\@nil\else
1717    \expandafter\main@language\expandafter{#1}%
1718  \fi}
1719 \def\bbl@provide@renew#1{%
1720  \ifx\bbl@KVP@captions\@nil\else
1721    \StartBabelCommands*{#1}{captions}%
1722      \bbl@read@ini{\bbl@KVP@captions}%   Here all letters cat = 11
1723      \bbl@after@ini
1724      \bbl@savestrings
1725    \EndBabelCommands
1726  \fi
1727  \ifx\bbl@KVP@import\@nil\else
1728    \StartBabelCommands*{#1}{date}%
1729      \bbl@savetoday
1730      \bbl@savedate
1731    \EndBabelCommands
1732  \fi
1733  \bbl@provide@hyphens{#1}}
```

The hyphenrules option is handled with an auxiliary macro.

```
1734 \def\bbl@provide@hyphens#1{%
1735  \let\bbl@tempa\relax
1736  \ifx\bbl@KVP@hyphenrules\@nil\else
1737    \bbl@replace\bbl@KVP@hyphenrules{ }{,}%
1738    \bbl@foreach\bbl@KVP@hyphenrules{%
1739      \ifx\bbl@tempa\relax    % if not yet found
1740        \bbl@ifsamestring{##1}{+}%
1741          {{\bbl@exp{\\\addlanguage\<l@##1>}}}%
1742          {}%
1743        \bbl@ifunset{l@##1}%
1744          {}%
1745          {\bbl@exp{\let\bbl@tempa\<l@##1>}}%
1746      \fi}%
1747  \fi
1748  \ifx\bbl@tempa\relax %         if no opt or no language in opt found
1749    \ifx\bbl@KVP@import\@nil\else % if importing
1750      \bbl@exp{%                  and hyphenrules is not empty
```

```
1751        \\\bbl@ifblank{\@nameuse{bbl@hyphr@#1}}%
1752          {}%
1753          {\let\\\bbl@tempa\<l@\@nameuse{bbl@hyphr@\languagename}>}}%
1754     \fi
1755   \fi
1756   \bbl@ifunset{bbl@tempa}%          ie, relax or undefined
1757     {\bbl@ifunset{l@#1}%            no hyphenrules found - fallback
1758        {\bbl@exp{\\\adddialect\<l@#1>\language}}%
1759        {}}%                         so, l@<lang> is ok - nothing to do
1760     {\bbl@exp{\\\adddialect\<l@#1>\bbl@tempa}}}%  found in opt list or ini
```

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. *TODO - Work in progress.*

```
1761 \def\bbl@read@ini#1{%
1762   \openin1=babel-#1.ini
1763   \ifeof1
1764     \bbl@error
1765       {There is no ini file for the requested language\\%
1766        (#1). Perhaps you misspelled it or your installation\\%
1767        is not complete.}%
1768       {Fix the name or reinstall babel.}%
1769   \else
1770     \let\bbl@section\@empty
1771     \let\bbl@savestrings\@empty
1772     \let\bbl@savetoday\@empty
1773     \let\bbl@savedate\@empty
1774     \let\bbl@inireader\bbl@iniskip
1775     \bbl@info{Importing data from babel-#1.ini for \languagename}%
1776     \loop
1777     \if T\ifeof1F\fi T\relax % Trick, because inside \loop
1778       \endlinechar\m@ne
1779       \read1 to \bbl@line
1780       \endlinechar`\^^M
1781       \ifx\bbl@line\@empty\else
1782         \expandafter\bbl@iniline\bbl@line\bbl@iniline
1783       \fi
1784     \repeat
1785   \fi}
1786 \def\bbl@iniline#1\bbl@iniline{%
1787   \@ifnextchar[\bbl@inisec{\@ifnextchar;\bbl@iniskip\bbl@inireader}#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 (TODO - but note the last section is not ended). By default, key=val pairs are ignored.

```
1788 \def\bbl@iniskip#1\@@{}%          if starts with ;
1789 \def\bbl@inisec[#1]#2\@@{%        if starts with opening bracket
1790   \@nameuse{bbl@secpost@\bbl@section}%  ends previous section
1791   \def\bbl@section{#1}%
1792   \@nameuse{bbl@secpre@\bbl@section}%    starts current section
1793   \bbl@ifunset{bbl@secline@#1}%
1794     {\let\bbl@inireader\bbl@iniskip}%
1795     {\bbl@exp{\let\\\bbl@inireader\<bbl@secline@#1>}}}
```

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

```
1796 \def\bbl@inikv#1=#2\@@{%          key=value
1797   \bbl@trim@def\bbl@tempa{#1}%
1798   \bbl@trim\toks@{#2}%
1799   \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.

```
1800 \def\bbl@exportkey#1#2#3{%
1801   \bbl@ifunset{bbl@@kv@#2}%
1802     {\bbl@csarg\gdef{#1@\languagename}{#3}}%
1803     {\expandafter\ifx\csname bbl@@kv@#2\endcsname\@empty
1804       \bbl@csarg\gdef{#1@\languagename}{#3}%
1805     \else
1806       \bbl@exp{\global\let\<bbl@#1@\languagename>\<bbl@@kv@#2>}%
1807     \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.

```
1808 \let\bbl@secline@identification\bbl@inikv
1809 \def\bbl@secpost@identification{%
1810   \bbl@exportkey{lname}{identification.name.english}{}%
1811   \bbl@exportkey{lbcp}{identification.tag.bcp47}{}%
1812   \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
1813   \bbl@exportkey{sname}{identification.script.name}{}%
1814   \bbl@exportkey{sbcp}{identification.script.tag.bcp47}{}%
1815   \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}}
1816 \let\bbl@secline@typography\bbl@inikv
1817 \def\bbl@after@ini{%
1818   \bbl@exportkey{lfthm}{typography.lefthyphenmin}{2}%
1819   \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
1820   \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
1821   \def\bbl@tempa{0.9}%
1822   \bbl@csarg\ifx{@kv@identification.version}\bbl@tempa
1823     \bbl@warning{%
1824       The `\languagename' date format may not be suitable\\%
1825       for proper typesetting, and therefore it very likely will\\%
1826       change in a future release. Reported}%
1827   \fi
1828   \bbl@toglobal\bbl@savetoday
1829   \bbl@toglobal\bbl@savedate}
```

Now captions and captions.licr, depending on the engine. And also for dates. They rely on a few auxilary macros.

```
1830 \ifcase\bbl@engine
1831   \bbl@csarg\def{secline@captions.licr}#1=#2\@@{%
1832     \bbl@ini@captions@aux{#1}{#2}}
1833   \bbl@csarg\def{secline@date.gregorian}#1=#2\@@{%        for defaults
1834     \bbl@ini@dategreg#1...\relax{#2}}
1835   \bbl@csarg\def{secline@date.gregorian.licr}#1=#2\@@{%  override
1836     \bbl@ini@dategreg#1...\relax{#2}}
1837 \else
1838   \def\bbl@secline@captions#1=#2\@@{%
1839     \bbl@ini@captions@aux{#1}{#2}}
1840   \bbl@csarg\def{secline@date.gregorian}#1=#2\@@{%
1841     \bbl@ini@dategreg#1...\relax{#2}}
1842 \fi
```

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

```
1843 \def\bbl@ini@captions@aux#1#2{%
1844   \bbl@trim@def\bbl@tempa{#1}%
1845   \bbl@ifblank{#2}%
1846     {\bbl@exp{%
1847       \toks@{\\\bbl@nocaption{\bbl@tempa}\<\languagename\bbl@tempa name>}}}%
```

```
1848        {\bbl@trim\toks@{#2}}%
1849    \bbl@exp{%
1850      \\\bbl@add\\\bbl@savestrings{%
1851        \\\SetString\<\bbl@tempa name>{\the\toks@}}}}
```

But dates are more complex. The full date format is stores in `date.gregorian`, so we must read it in non-Unicode engines, too.

```
1852 \bbl@csarg\def{secpre@date.gregorian.licr}{%
1853    \ifcase\bbl@engine\let\bbl@savedate\@empty\fi}
1854 \def\bbl@ini@dategreg#1.#2.#3.#4\relax#5{% TODO - ignore with 'captions'
1855    \bbl@trim@def\bbl@tempa{#1.#2}%
1856    \bbl@ifsamestring{\bbl@tempa}{months.wide}%
1857      {\bbl@trim@def\bbl@tempa{#3}%
1858       \bbl@trim\toks@{#5}%
1859       \bbl@exp{%
1860        \\\bbl@add\\\bbl@savedate{%
1861          \\\SetString\<month\romannumeral\bbl@tempa name>{\the\toks@}}}}%
1862      {\bbl@ifsamestring{\bbl@tempa}{date.long}%
1863        {\bbl@trim@def\bbl@toreplace{#5}%
1864         \bbl@TG@@date
1865         \global\bbl@csarg\let{date@\languagename}\bbl@toreplace
1866         \bbl@exp{%
1867          \gdef\<\languagename date>{\\\protect\<\languagename date >}%
1868          \gdef\<\languagename date >####1####2####3{%
1869            \\\bbl@usedategrouptrue
1870            \<bbl@ensure@\languagename>{%
1871              \<bbl@date@\languagename>{####1}{####2}{####3}}%
1872          \\\bbl@add\\\bbl@savetoday{%
1873            \\\SetString\\\today{%
1874              \<\languagename date>{\\\the\year}{\\\the\month}{\\\the\day}}}}}}%
1875        {}}
```

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" inconsistenly in either in the date or in the month name.

```
1876 \newcommand\BabelDateSpace{\nobreakspace}
1877 \newcommand\BabelDateDot{.\@}
1878 \newcommand\BabelDated[1]{{\number#1}}
1879 \newcommand\BabelDatedd[1]{{\ifnum#1<10 0\fi\number#1}}
1880 \newcommand\BabelDateM[1]{{\number#1}}
1881 \newcommand\BabelDateMM[1]{{\ifnum#1<10 0\fi\number#1}}
1882 \newcommand\BabelDateMMMM[1]{{%
1883    \csname month\romannumeral#1name\endcsname}}%
1884 \newcommand\BabelDatey[1]{{\number#1}}%
1885 \newcommand\BabelDateyy[1]{{%
1886    \ifnum#1<10 0\number#1 %
1887    \else\ifnum#1<100 \number#1 %
1888    \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
1889    \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
1890    \else
1891      \bbl@error
1892        {Currently two-digit years are restricted to the\\
1893         range 0-9999.}%
1894        {There is little you can do. Sorry.}%
1895    \fi\fi\fi\fi}}
1896 \newcommand\BabelDateyyyy[1]{{\number#1}}
1897 \def\bbl@replace@finish@iii#1{%
1898    \bbl@exp{\def\\#1####1####2####3{\the\toks@}}}
1899 \def\bbl@TG@@date{%
```

97

```
1900    \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace{}}%
1901    \bbl@replace\bbl@toreplace{[.]}{\BabelDateDot{}}%
1902    \bbl@replace\bbl@toreplace{[d]}{\BabelDated{####3}}%
1903    \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{####3}}%
1904    \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{####2}}%
1905    \bbl@replace\bbl@toreplace{[MM]}{\BabelDateMM{####2}}%
1906    \bbl@replace\bbl@toreplace{[MMMM]}{\BabelDateMMMM{####2}}%
1907    \bbl@replace\bbl@toreplace{[y]}{\BabelDatey{####1}}%
1908    \bbl@replace\bbl@toreplace{[yy]}{\BabelDateyy{####1}}%
1909    \bbl@replace\bbl@toreplace{[yyyy]}{\BabelDateyyyy{####1}}%
1910 % Note after \bbl@replace \toks@ contains the resulting string.
1911 % TODO - Using this implicit behavior doesn't seem a good idea.
1912    \bbl@replace@finish@iii\bbl@toreplace}
```

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

```
1913 \def\bbl@provide@lsys#1{%
1914   \bbl@ifunset{bbl@lname@#1}%
1915     {\bbl@ini@ids{#1}}%
1916     {}%
1917   \bbl@csarg\let{lsys@#1}\@empty
1918   \bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}{}%
1919   \bbl@ifunset{bbl@sotf#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{}%
1920   \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
1921   \bbl@ifunset{bbl@lname@#1}{}%
1922     {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
1923   \bbl@csarg\bbl@toglobal{lsys@#1}}%
1924 %  \bbl@exp{% TODO - should be global
1925 %    \<keys_if_exist:nnF>{fontspec-opentype/Script}{\bbl@cs{sname@#1}}%
1926 %      {\\\newfontscript{\bbl@cs{sname@#1}}{\bbl@cs{sotf@#1}}}%
1927 %    \<keys_if_exist:nnF>{fontspec-opentype/Language}{\bbl@cs{lname@#1}}%
1928 %      {\\\newfontlanguage{\bbl@cs{lname@#1}}{\bbl@cs{lotf@#1}}}}}
```

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.

```
1929 \def\bbl@ini@ids#1{%
1930   \def\BabelBeforeIni##1##2{%
1931     \begingroup
1932       \bbl@add\bbl@secpost@identification{\closein1 }%
1933       \catcode`\[=12 \catcode`\]=12 \catcode`\==12
1934       \bbl@read@ini{##1}%
1935     \endgroup}%            boxed, to avoid extra spaces:
1936   {\setbox\z@\hbox{\InputIfFileExists{babel-#1.tex}{}{}}}}
```

# 9    The kernel of Babel (`babel.def`, only LaTeX)

## 9.1    The redefinition of the style commands

The rest of the code in this file can only be processed by LaTeX, so we check the current format. If it is plain TeX, processing should stop here. But, because of the need to limit the scope of the definition of \format, a macro that is used locally in the following \if statement, this comparison is done inside a group. To prevent TeX from complaining about an unclosed group, the processing of the command \endinput is deferred until after the group is closed. This is accomplished by the command \aftergroup.

```
1937 {\def\format{lplain}
1938 \ifx\fmtname\format
1939 \else
1940   \def\format{LaTeX2e}
1941   \ifx\fmtname\format
1942   \else
1943     \aftergroup\endinput
1944   \fi
1945 \fi}
```

## 9.2  Cross referencing macros

The LATEX book states:

> The *key* argument is any sequence of letters, digits, and punctuation symbols; upper-
> and lowercase letters are regarded as different.

When the above quote should still be true when a document is typeset in a language that
has active characters, special care has to be taken of the category codes of these characters
when they appear in an argument of the cross referencing macros.
When a cross referencing command processes its argument, all tokens in this argument
should be character tokens with category 'letter' or 'other'.
The only way to accomplish this in most cases is to use the trick described in the
TEXbook [2] (Appendix D, page 382). The primitive \meaning applied to a token expands to
the current meaning of this token. For example, '\meaning\A' with \A defined as
'\def\A#1{\B}' expands to the characters 'macro:#1->\B' with all category codes set to
'other' or 'space'.

\newlabel   The macro \label writes a line with a \newlabel command into the .aux file to define
labels.

```
1946 %\bbl@redefine\newlabel#1#2{%
1947 %    \@safe@activestrue\org@newlabel{#1}{#2}\@safe@activesfalse}
```

\@newl@bel   We need to change the definition of the LATEX-internal macro \@newl@bel. This is needed
because we need to make sure that shorthand characters expand to their non-active
version.
The following package options control which macros are to be redefined.

```
1948 ⟨⟨∗More package options⟩⟩ ≡
1949 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
1950 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
1951 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
1952 ⟨⟨/More package options⟩⟩
```

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.

```
1953 \bbl@trace{Cross referencing macros}
1954 \ifx\bbl@opt@safe\@empty\else
1955   \def\@newl@bel#1#2#3{%
1956     {\@safe@activestrue
1957     \bbl@ifunset{#1@#2}%
1958        \relax
1959        {\gdef\@multiplelabels{%
1960           \@latex@warning@no@line{There were multiply-defined labels}}%
1961        \@latex@warning@no@line{Label `#2' multiply defined}}%
1962     \global\@namedef{#1@#2}{#3}}}
```

\@testdef An internal LaTeX macro used to test if the labels that have been written on the `.aux` file have changed. It is called by the \enddocument macro. This macro needs to be completely rewritten, using \meaning. The reason for this is that in some cases the expansion of \#1@#2 contains the same characters as the #3; but the character codes differ. Therefore LaTeX keeps reporting that the labels may have changed.

```
1963 \CheckCommand*\@testdef[3]{%
1964   \def\reserved@a{#3}%
1965   \expandafter\ifx\csname#1@#2\endcsname\reserved@a
1966   \else
1967     \@tempswatrue
1968   \fi}
```

Now that we made sure that \@testdef still has the same definition we can rewrite it. First we make the shorthands 'safe'.

```
1969 \def\@testdef#1#2#3{%
1970   \@safe@activestrue
```

Then we use \bbl@tempa as an 'alias' for the macro that contains the label which is being checked.

```
1971   \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
```

Then we define \bbl@tempb just as \@newl@bel does it.

```
1972   \def\bbl@tempb{#3}%
1973   \@safe@activesfalse
```

When the label is defined we replace the definition of \bbl@tempa by its meaning.

```
1974   \ifx\bbl@tempa\relax
1975   \else
1976     \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
1977   \fi
```

We do the same for \bbl@tempb.

```
1978   \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
```

If the label didn't change, \bbl@tempa and \bbl@tempb should be identical macros.

```
1979   \ifx\bbl@tempa\bbl@tempb
1980   \else
1981     \@tempswatrue
1982   \fi}
1983 \fi
```

\ref  The same holds for the macro \ref that references a label and \pageref to reference a
\pageref page. So we redefine \ref and \pageref. While we change these macros, we make them robust as well (if they weren't already) to prevent problems if they should become expanded at the wrong moment.

```
1984 \bbl@xin@{R}\bbl@opt@safe
1985 \ifin@
1986   \bbl@redefinerobust\ref#1{%
1987     \@safe@activestrue\org@ref{#1}\@safe@activesfalse}
1988   \bbl@redefinerobust\pageref#1{%
1989     \@safe@activestrue\org@pageref{#1}\@safe@activesfalse}
1990 \else
1991   \let\org@ref\ref
1992   \let\org@pageref\pageref
1993 \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.

```
1994 \bbl@xin@{B}\bbl@opt@safe
1995 \ifin@
1996   \bbl@redefine\@citex[#1]#2{%
1997     \@safe@activestrue\edef\@tempa{#2}\@safe@activesfalse
1998     \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.

```
1999   \AtBeginDocument{%
2000     \@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.)

```
2001     \def\@citex[#1][#2]#3{%
2002       \@safe@activestrue\edef\@tempa{#3}\@safe@activesfalse
2003       \org@@citex[#1][#2]{\@tempa}}%
2004     }{}}
```

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

```
2005   \AtBeginDocument{%
2006     \@ifpackageloaded{cite}{%
2007       \def\@citex[#1]#2{%
2008         \@safe@activestrue\org@@citex[#1]{#2}\@safe@activesfalse}%
2009       }{}}
```

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

```
2010 \bbl@redefine\nocite#1{%
2011   \@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.

```
2012 \bbl@redefine\bibcite{%
```

We call \bbl@cite@choice to select the proper definition for \bibcite. This new definition is then activated.

```
2013   \bbl@cite@choice
2014   \bibcite}
```

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

```
2015 \def\bbl@bibcite#1#2{%
2016   \org@bibcite{#1}{\@safe@activesfalse#2}}
```

**\bbl@cite@choice**  The macro \bbl@cite@choice determines which definition of \bibcite is needed.

```
2017 \def\bbl@cite@choice{%
```

First we give \bibcite its default definition.

```
2018     \global\let\bibcite\bbl@bibcite
```

Then, when natbib is loaded we restore the original definition of \bibcite.

```
2019     \@ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{}%
```

For cite we do the same.

```
2020     \@ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{}%
```

Make sure this only happens once.

```
2021     \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.

```
2022     \AtBeginDocument{\bbl@cite@choice}
```

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

```
2023 \bbl@redefine\@bibitem#1{%
2024     \@safe@activestrue\org@@bibitem{#1}\@safe@activesfalse}
2025 \else
2026   \let\org@nocite\nocite
2027   \let\org@@citex\@citex
2028   \let\org@bibcite\bibcite
2029   \let\org@@bibitem\@bibitem
2030 \fi
```

## 9.3   Marks

**\markright**  Because the output routine is asynchronous, we must pass the current language attribute to the head lines, together with the text that is put into them. To achieve this we need to adapt the definition of \markright and \markboth somewhat.
We check whether the argument is empty; if it is, we just make sure the scratch token register is empty. Next, we store the argument to \markright in the scratch token register. This way these commands will not be expanded later, and we make sure that the text is typeset using the correct language settings. While doing so, we make sure that active characters that may end up in the mark are not disabled by the output routine kicking in while \@safe@activestrue is in effect.

```
2031 \bbl@trace{Marks}
2032 \IfBabelLayout{sectioning}
2033   {\ifx\bbl@opt@headfoot\@nnil
2034     \g@addto@macro\@resetactivechars{%
2035       \set@typeset@protect
2036       \expandafter\select@language@x\expandafter{\bbl@main@language}%
2037       \let\protect\noexpand}%
2038   \fi}
2039   {\bbl@redefine\markright#1{%
2040     \bbl@ifblank{#1}%
2041       {\org@markright{}}%
2042       {\toks@{#1}%
2043         \bbl@exp{%
2044           \\\org@markright{\\\protect\\\foreignlanguage{\languagename}%
2045             {\\\protect\\\bbl@restore@actives\the\toks@}}}}}}
```

102

\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.

```
2046    \ifx\@mkboth\markboth
2047      \def\bbl@tempc{\let\@mkboth\markboth}
2048    \else
2049      \def\bbl@tempc{}
2050    \fi
```

Now we can start the new definition of \markboth

```
2051    \bbl@redefine\markboth#1#2{%
2052      \protected@edef\bbl@tempb##1{%
2053        \protect\foreignlanguage{\languagename}{\protect\bbl@restore@actives##1}}%
2054      \bbl@ifblank{#1}%
2055        {\toks@{}}%
2056        {\toks@\expandafter{\bbl@tempb{#1}}}%
2057      \bbl@ifblank{#2}%
2058        {\@temptokena{}}%
2059        {\@temptokena\expandafter{\bbl@tempb{#2}}}%
2060      \bbl@exp{\\\org@markboth{\the\toks@}{\the\@temptokena}}
```

and copy it to \@mkboth if necessary.

```
2061    \bbl@tempc}  % end \IfBabelLayout
```

## 9.4  Preventing clashes with other packages

### 9.4.1  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.
The first thing we need to do is check if the package ifthen is loaded. This should be done
at \begin{document} time.

```
2062 \bbl@trace{Preventing clashes with other packages}
2063 \bbl@xin@{R}\bbl@opt@safe
2064 \ifin@
2065   \AtBeginDocument{%
2066     \@ifpackageloaded{ifthen}{%
```

Then we can redefine \ifthenelse:

```
2067        \bbl@redefine@long\ifthenelse#1#2#3{%
```

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.

```
2068          \let\bbl@temp@pref\pageref
2069          \let\pageref\org@pageref
2070          \let\bbl@temp@ref\ref
2071          \let\ref\org@ref
```

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. When the package wasn't loaded we do nothing.

```
2072        \@safe@activestrue
2073        \org@ifthenelse{#1}%
2074          {\let\pageref\bbl@temp@pref
2075           \let\ref\bbl@temp@ref
2076           \@safe@activesfalse
2077           #2}%
2078          {\let\pageref\bbl@temp@pref
2079           \let\ref\bbl@temp@ref
2080           \@safe@activesfalse
2081           #3}%
2082        }%
2083      }{}%
2084    }
```

### 9.4.2  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.

```
2085  \AtBeginDocument{%
2086    \@ifpackageloaded{varioref}{%
2087      \bbl@redefine\@@vpageref#1[#2]#3{%
2088        \@safe@activestrue
2089        \org@@@vpageref{#1}[#2]{#3}%
2090        \@safe@activesfalse}%
```

The same needs to happen for \vrefpagenum.

```
2091      \bbl@redefine\vrefpagenum#1#2{%
2092        \@safe@activestrue
2093        \org@vrefpagenum{#1}{#2}%
2094        \@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 exandable form of \ref. So we employ a little trick here. We redefine the (internal) command \Ref␣ to call \org@ref instead of \ref. The disadvantgage of this solution is that whenever the derfinition of \Ref changes, this definition needs to be updated as well.

```
2095      \expandafter\def\csname Ref \endcsname#1{%
2096        \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
2097      }{}%
2098    }
2099  \fi
```

### 9.4.3  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.

So at \begin{document} we check whether hhline is loaded.

```
2100 \AtEndOfPackage{%
2101   \AtBeginDocument{%
2102     \@ifpackageloaded{hhline}%
```

104

Then we check whether the expansion of \normal@char: is not equal to \relax.

```
2103        {\expandafter\ifx\csname normal@char\string:\endcsname\relax
2104         \else
```

In that case we simply reload the package. 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.

```
2105          \makeatletter
2106          \def\@currname{hhline}\input{hhline.sty}\makeatother
2107        \fi}%
2108      {}}}
```

### 9.4.4 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.

```
2109 \AtBeginDocument{%
2110   \ifx\pdfstringdefDisableCommands\@undefined\else
2111     \pdfstringdefDisableCommands{\languageshorthands{system}}%
2112   \fi}
```

### 9.4.5 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.

```
2113 \DeclareRobustCommand{\FOREIGNLANGUAGE}[1]{%
2114   \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.

```
2115 \def\substitutefontfamily#1#2#3{%
2116   \lowercase{\immediate\openout15=#1#2.fd\relax}%
2117   \immediate\write15{%
2118     \string\ProvidesFile{#1#2.fd}%
2119     [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
2120      \space generated font description file]^^J
2121     \string\DeclareFontFamily{#1}{#2}{}^^J
2122     \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{}^^J
2123     \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{}^^J
2124     \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{}^^J
2125     \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{}^^J
2126     \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{}^^J
2127     \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{}^^J
2128     \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{}^^J
2129     \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{}^^J
2130   }%
2131   \closeout15
2132   }
```

This command should only be used in the preamble of a document.

```
2133 \@onlypreamble\substitutefontfamily
```

## 9.5 Encoding and fonts

Because documents may use non-ASCII font encodings, we make sure that the logos of TeX and LaTeX always come out in the right encoding. There is a list of non-ASCII encodings. Unfortunately, fontenc deletes its package options, so we must guess which encodings has been loaded by traversing \@filelist to search for ⟨*enc*⟩enc.def. If a non-ASCII has been loaded, we define versions of \TeX and \LaTeX for them using \ensureascii. The default ASCII encoding is set, too (in reverse order): the "main" encoding (when the document begins), the last loaded, or OT1.

\ensureascii

```
2134 \bbl@trace{Encoding and fonts}
2135 \newcommand\BabelNonASCII{LGR,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU,}
2136 \let\org@TeX\TeX
2137 \let\org@LaTeX\LaTeX
2138 \let\ensureascii\@firstofone
2139 \AtBeginDocument{%
2140   \in@false
2141   \bbl@foreach\BabelNonASCII{% is there a non-ascii enc?
2142     \ifin@\else
2143       \lowercase{\bbl@xin@{,#1enc.def,}{,\@filelist,}}%
2144     \fi}%
2145   \ifin@ % if a non-ascii has been loaded
2146     \def\ensureascii#1{{\fontencoding{OT1}\selectfont#1}}%
2147     \DeclareTextCommandDefault{\TeX}{\org@TeX}%
2148     \DeclareTextCommandDefault{\LaTeX}{\org@LaTeX}%
2149     \def\bbl@tempb#1\@@{\uppercase{\bbl@tempc#1}ENC.DEF\@empty\@@}%
2150     \def\bbl@tempc#1ENC.DEF#2\@@{%
2151       \ifx\@empty#2\else
2152         \bbl@ifunset{T@#1}%
2153           {}%
2154           {\bbl@xin@{,#1,}{,\BabelNonASCII,}%
2155            \ifin@
2156              \DeclareTextCommand{\TeX}{#1}{\ensureascii{\org@TeX}}%
2157              \DeclareTextCommand{\LaTeX}{#1}{\ensureascii{\org@LaTeX}}%
2158            \else
2159              \def\ensureascii##1{{\fontencoding{#1}\selectfont##1}}%
2160            \fi}%
2161       \fi}%
2162     \bbl@foreach\@filelist{\bbl@tempb#1\@@}%  TODO - \@@ de mas??
2163     \bbl@xin@{,\cf@encoding,}{,\BabelNonASCII,}%
2164     \ifin@\else
2165       \edef\ensureascii#1{{%
2166         \noexpand\fontencoding{\cf@encoding}\noexpand\selectfont#1}}%
2167     \fi
2168   \fi}
```

Now comes the old deprecated stuff (with a little change in 3.9l, for fontspec). The first thing we need to do is to determine, at \begin{document}, which latin fontencoding to use.

\latinencoding    When text is being typeset in an encoding other than 'latin' (OT1 or T1), it would be nice to still have Roman numerals come out in the Latin encoding. So we first assume that the current encoding at the end of processing the package is the Latin encoding.

```
2169 \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

106

have to revert to parsing the internal macro `\@filelist` which contains all the filenames
loaded.

```
2170 \AtBeginDocument{%
2171   \@ifpackageloaded{fontspec}%
2172     {\xdef\latinencoding{%
2173        \ifx\UTFencname\@undefined
2174          EU\ifcase\bbl@engine\or2\or1\fi
2175        \else
2176          \UTFencname
2177        \fi}}%
2178     {\gdef\latinencoding{OT1}%
2179     \ifx\cf@encoding\bbl@t@one
2180       \xdef\latinencoding{\bbl@t@one}%
2181     \else
2182       \@ifl@aded{def}{t1enc}{\xdef\latinencoding{\bbl@t@one}}{}%
2183     \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.

```
2184 \DeclareRobustCommand{\latintext}{%
2185   \fontencoding{\latinencoding}\selectfont
2186   \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.

```
2187 \ifx\@undefined\DeclareTextFontCommand
2188   \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
2189 \else
2190   \DeclareTextFontCommand{\textlatin}{\latintext}
2191 \fi
```

## 9.6 Basic bidi support

**Work in progress.** This code is currently placed here for practical reasons.
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 posible, too. Its
  main drawback is font handling is often considered to be less mature than xetex,

107

mainly in Indic scripts (but there are steps to make HarfBuzz, the xetex font engine, available in luatex; see <https://github.com/tatzetwerk/luatex-harfbuzz>).

```
2192 \bbl@trace{Basic (internal) bidi support}
2193 \def\bbl@alscripts{,Arabic,Syriac,Thaana,}
2194 \def\bbl@rscripts{%
2195   ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
2196   Old Hungarian,Old Hungarian,Lydian,Mandaean,Manichaean,%
2197   Manichaean,Meroitic Cursive,Meroitic,Old North Arabian,%
2198   Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
2199   Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
2200   Old South Arabian,}%
2201 \def\bbl@provide@dirs#1{%
2202   \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
2203   \ifin@
2204     \global\bbl@csarg\chardef{wdir@#1}\@ne
2205     \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
2206     \ifin@
2207       \global\bbl@csarg\chardef{wdir@#1}\tw@  % useless in xetex
2208     \fi
2209   \else
2210     \global\bbl@csarg\chardef{wdir@#1}\z@
2211   \fi}
2212 \def\bbl@switchdir{%
2213   \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
2214   \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
2215   \bbl@exp{\\\bbl@setdirs\bbl@cs{wdir@\languagename}}}
2216 \def\bbl@setdirs#1{% TODO - math
2217   \ifcase\bbl@select@type % TODO - strictly, not the right test
2218     \bbl@bodydir{#1}%
2219     \bbl@pardir{#1}%
2220   \fi
2221   \bbl@textdir{#1}}
2222 \ifodd\bbl@engine  % luatex=1
2223   \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
2224   \DisableBabelHook{babel-bidi}
2225   \chardef\bbl@thepardir\z@
2226   \def\bbl@getluadir#1{%
2227     \directlua{
2228       if tex.#1dir == 'TLT' then
2229         tex.sprint('0')
2230       elseif tex.#1dir == 'TRT' then
2231         tex.sprint('1')
2232       end}}
2233   \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
2234     \ifcase#3\relax
2235       \ifcase\bbl@getluadir{#1}\relax\else
2236         #2 TLT\relax
2237       \fi
2238     \else
2239       \ifcase\bbl@getluadir{#1}\relax
2240         #2 TRT\relax
2241       \fi
2242     \fi}
2243   \def\bbl@textdir#1{%
2244     \bbl@setluadir{text}\textdir{#1}% TODO - ?\linedir
2245     \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*3+#1}}
2246   \def\bbl@pardir#1{\bbl@setluadir{par}\pardir{#1}%
2247     \chardef\bbl@thepardir#1\relax}
```

```
2248    \def\bbl@bodydir{\bbl@setluadir{body}\bodydir}
2249    \def\bbl@pagedir{\bbl@setluadir{page}\pagedir}
2250    \def\bbl@dirparastext{\pardir\the\textdir\relax}%    %%%%
2251 \else % pdftex=0, xetex=2
2252    \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
2253    \DisableBabelHook{babel-bidi}
2254    \newcount\bbl@dirlevel
2255    \chardef\bbl@thetextdir\z@
2256    \chardef\bbl@thepardir\z@
2257    \def\bbl@textdir#1{%
2258      \ifcase#1\relax
2259        \chardef\bbl@thetextdir\z@
2260        \bbl@textdir@i\beginL\endL
2261      \else
2262        \chardef\bbl@thetextdir\@ne
2263        \bbl@textdir@i\beginR\endR
2264      \fi}
2265    \def\bbl@textdir@i#1#2{%
2266      \ifhmode
2267        \ifnum\currentgrouplevel>\z@
2268          \ifnum\currentgrouplevel=\bbl@dirlevel
2269            \bbl@error{Multiple bidi settings inside a group}%
2270              {I'll insert a new group, but expect wrong results.}%
2271            \bgroup\aftergroup#2\aftergroup\egroup
2272          \else
2273            \ifcase\currentgrouptype\or % 0 bottom
2274              \aftergroup#2% 1 simple {}
2275            \or
2276              \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
2277            \or
2278              \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
2279            \or\or\or % vbox vtop align
2280            \or
2281              \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
2282            \or\or\or\or\or\or % output math disc insert vcent mathchoice
2283            \or
2284              \aftergroup#2% 14 \begingroup
2285            \else
2286              \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
2287            \fi
2288          \fi
2289          \bbl@dirlevel\currentgrouplevel
2290        \fi
2291        #1%
2292      \fi}
2293    \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
2294    \let\bbl@bodydir\@gobble
2295    \let\bbl@pagedir\@gobble
2296    \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).

```
2297    \def\bbl@xebidipar{%
2298      \let\bbl@xebidipar\relax
2299      \TeXXeTstate\@ne
2300      \def\bbl@xeeverypar{%
2301        \ifcase\bbl@thepardir
2302          \ifcase\bbl@thetextdir\else\beginR\fi
```

```
2303        \else
2304          {\setbox\z@\lastbox\beginR\box\z@}%
2305        \fi}%
2306      \let\bbl@severypar\everypar
2307      \newtoks\everypar
2308      \everypar=\bbl@severypar
2309      \bbl@severypar{\bbl@xeeverypar\the\everypar}}}
2310 \fi
```

A tool for weak L (mainly digits).

```
2311 \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}
```

### 9.7  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.

```
2312 \bbl@trace{Local Language Configuration}
2313 \ifx\loadlocalcfg\@undefined
2314   \@ifpackagewith{babel}{noconfigs}%
2315     {\let\loadlocalcfg\@gobble}%
2316     {\def\loadlocalcfg#1{%
2317       \InputIfFileExists{#1.cfg}%
2318         {\typeout{*************************************^^J%
2319                     * Local config file #1.cfg used^^J%
2320                     *}}%
2321       \@empty}}
2322 \fi
```

Just to be compatible with LaTeX 2.09 we add a few more lines of code:

```
2323 \ifx\@unexpandable@protect\@undefined
2324   \def\@unexpandable@protect{\noexpand\protect\noexpand}
2325   \long\def\protected@write#1#2#3{%
2326     \begingroup
2327       \let\thepage\relax
2328       #2%
2329       \let\protect\@unexpandable@protect
2330       \edef\reserved@a{\write#1{#3}}%
2331       \reserved@a
2332     \endgroup
2333     \if@nobreak\ifvmode\nobreak\fi\fi}
2334 \fi
2335 ⟨/core⟩
2336 ⟨*kernel⟩
```

## 10  Multiple languages (switch.def)

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.

```
2337 ⟨⟨Make sure ProvidesFile is defined⟩⟩
2338 \ProvidesFile{switch.def}[⟨⟨date⟩⟩ ⟨⟨version⟩⟩ Babel switching mechanism]
2339 ⟨⟨Load macros for plain if not LaTeX⟩⟩
```

110

2340 ⟨⟨*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.

```
2341 \def\bbl@version{⟨⟨version⟩⟩}
2342 \def\bbl@date{⟨⟨date⟩⟩}
2343 \def\adddialect#1#2{%
2344   \global\chardef#1#2\relax
2345   \bbl@usehooks{adddialect}{{#1}{#2}}%
2346   \wlog{\string#1 = a dialect from \string\language#2}}
```

\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.

```
2347 \def\bbl@fixname#1{%
2348   \begingroup
2349     \def\bbl@tempe{l@}%
2350     \edef\bbl@tempd{\noexpand\@ifundefined{\noexpand\bbl@tempe#1}}%
2351     \bbl@tempd
2352       {\lowercase\expandafter{\bbl@tempd}%
2353         {\uppercase\expandafter{\bbl@tempd}%
2354           \@empty
2355           {\edef\bbl@tempd{\def\noexpand#1{#1}}%
2356            \uppercase\expandafter{\bbl@tempd}}}%
2357       {\edef\bbl@tempd{\def\noexpand#1{#1}}%
2358        \lowercase\expandafter{\bbl@tempd}}}%
2359     \@empty
2360     \edef\bbl@tempd{\endgroup\def\noexpand#1{#1}}%
2361   \bbl@tempd}
2362 \def\bbl@iflanguage#1{%
2363   \@ifundefined{l@#1}{\@nolanerr{#1}\@gobble}\@firstofone}
```

\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.

```
2364 \def\iflanguage#1{%
2365   \bbl@iflanguage{#1}{%
2366     \ifnum\csname l@#1\endcsname=\language
2367       \expandafter\@firstoftwo
2368     \else
2369       \expandafter\@secondoftwo
2370     \fi}}
```

## 10.1   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.

To allow the call of \selectlanguage either with a control sequence name or with a simple string as argument, we have to use a trick to delete the optional escape character.

To convert a control sequence to a string, we use the \string primitive. Next we have to look at the first character of this string and compare it with the escape character. Because this escape character can be changed by setting the internal integer \escapechar to a character number, we have to compare this number with the character of the string. To do this we have to use TEX's backquote notation to specify the character as a number.

If the first character of the \string'ed argument is the current escape character, the comparison has stripped this character and the rest in the 'then' part consists of the rest of the control sequence name. Otherwise we know that either the argument is not a control sequence or \escapechar is set to a value outside of the character range 0–255.

If the user gives an empty argument, we provide a default argument for \string. This argument should expand to nothing.

```
2371 \let\bbl@select@type\z@
2372 \edef\selectlanguage{%
2373    \noexpand\protect
2374    \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.

```
2375 \ifx\@undefined\protect\let\protect\relax\fi
```

As LATEX 2.09 writes to files *expanded* whereas LATEX 2ε takes care *not* to expand the arguments of \write statements we need to be a bit clever about the way we add information to .aux files. Therefore we introduce the macro \xstring which should expand to the right amount of \string's.

```
2376 \ifx\documentclass\@undefined
2377    \def\xstring{\string\string\string}
2378 \else
2379    \let\xstring\string
2380 \fi
```

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.

```
2381 \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:

```
2382 \def\bbl@push@language{%
2383    \xdef\bbl@language@stack{\languagename+\bbl@language@stack}}
```

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 (delimited by '-') in its third argument.

```
2384 \def\bbl@pop@lang#1+#2-#3{%
2385   \edef\languagename{#1}\xdef#3{#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) followed by the '-'-sign and finally the reference to the stack.

```
2386 \let\bbl@ifrestoring\@secondoftwo
2387 \def\bbl@pop@language{%
2388   \expandafter\bbl@pop@lang\bbl@language@stack-\bbl@language@stack
2389   \let\bbl@ifrestoring\@firstoftwo
2390   \expandafter\bbl@set@language\expandafter{\languagename}%
2391   \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.

```
2392 \expandafter\def\csname selectlanguage \endcsname#1{%
2393   \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\tw@\fi
2394   \bbl@push@language
2395   \aftergroup\bbl@pop@language
2396   \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 not well defined. 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.

```
2397 \def\BabelContentsFiles{toc,lof,lot}
2398 \def\bbl@set@language#1{%
2399   \edef\languagename{%
2400     \ifnum\escapechar=\expandafter`\string#1\@empty
2401     \else\string#1\@empty\fi}%
2402   \select@language{\languagename}%
2403   \expandafter\ifx\csname date\languagename\endcsname\relax\else
2404     \if@filesw
2405       \protected@write\@auxout{}{\string\babel@aux{\languagename}{}}%
2406       \bbl@usehooks{write}{}%
2407     \fi
2408   \fi}
2409 \def\select@language#1{%
2410   \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
2411   \edef\languagename{#1}%
2412   \bbl@fixname\languagename
2413   \bbl@iflanguage\languagename{%
2414     \expandafter\ifx\csname date\languagename\endcsname\relax
2415       \bbl@error
2416         {Unknown language `#1'. Either you have\\%
2417          misspelled its name, it has not been installed,\\%
```

```
2418        or you requested it in a previous run. Fix its name,\\%
2419        install it or just rerun the file, respectively}%
2420       {You may proceed, but expect wrong results}%
2421     \else
2422       \let\bbl@select@type\z@
2423       \expandafter\bbl@switch\expandafter{\languagename}%
2424     \fi}}
2425 \def\babel@aux#1#2{%
2426   \select@language{#1}%
2427   \bbl@foreach\BabelContentsFiles{%
2428     \@writefile{##1}{\babel@toc{#1}{#2}}}}} %% TODO - ok in plain?
2429 \def\babel@toc#1#2{%
2430   \select@language{#1}}
```

A bit of optimization. Select in heads/foots the language only if necessary. The real thing is in `babel.def`.

```
2431 \let\select@language@x\select@language
```

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.

```
2432 \newif\ifbbl@usedategroup
2433 \def\bbl@switch#1{%
2434   \originalTeX
2435   \expandafter\def\expandafter\originalTeX\expandafter{%
2436     \csname noextras#1\endcsname
2437     \let\originalTeX\@empty
2438     \babel@beginsave}%
2439   \bbl@usehooks{afterreset}{}%
2440   \languageshorthands{none}%
2441   \ifcase\bbl@select@type
2442     \ifhmode
2443       \hskip\z@skip % trick to ignore spaces
2444       \csname captions#1\endcsname\relax
2445       \csname date#1\endcsname\relax
2446       \loop\ifdim\lastskip>\z@\unskip\repeat\unskip
2447     \else
2448       \csname captions#1\endcsname\relax
2449       \csname date#1\endcsname\relax
2450     \fi
2451   \else\ifbbl@usedategroup
2452     \bbl@usedategroupfalse
2453     \ifhmode
2454       \hskip\z@skip % trick to ignore spaces
2455       \csname date#1\endcsname\relax
2456       \loop\ifdim\lastskip>\z@\unskip\repeat\unskip
```

```
2457        \else
2458          \csname date#1\endcsname\relax
2459        \fi
2460      \fi\fi
2461      \bbl@usehooks{beforeextras}{}%
2462      \csname extras#1\endcsname\relax
2463      \bbl@usehooks{afterextras}{}%
2464      \ifcase\bbl@opt@hyphenmap\or
2465        \def\BabelLower##1##2{\lccode##1=##2\relax}%
2466        \ifnum\bbl@hymapsel>4\else
2467          \csname\languagename @bbl@hyphenmap\endcsname
2468        \fi
2469        \chardef\bbl@opt@hyphenmap\z@
2470      \else
2471        \ifnum\bbl@hymapsel>\bbl@opt@hyphenmap\else
2472          \csname\languagename @bbl@hyphenmap\endcsname
2473        \fi
2474      \fi
2475      \global\let\bbl@hymapsel\@cclv
2476      \bbl@patterns{#1}%
2477      \babel@savevariable\lefthyphenmin
2478      \babel@savevariable\righthyphenmin
2479      \expandafter\ifx\csname #1hyphenmins\endcsname\relax
2480        \set@hyphenmins\tw@\thr@@\relax
2481      \else
2482        \expandafter\expandafter\expandafter\set@hyphenmins
2483          \csname #1hyphenmins\endcsname\relax
2484      \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.

```
2485 \long\def\otherlanguage#1{%
2486    \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\thr@@\fi
2487    \csname selectlanguage \endcsname{#1}%
2488    \ignorespaces}
```

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

```
2489 \long\def\endotherlanguage{%
2490    \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.

```
2491 \expandafter\def\csname otherlanguage*\endcsname#1{%
2492    \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
2493    \foreign@language{#1}}
```

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".

```
2494 \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.

```
2495 \providecommand\bbl@beforeforeign{}
2496 \edef\foreignlanguage{%
2497   \noexpand\protect
2498   \expandafter\noexpand\csname foreignlanguage \endcsname}
2499 \expandafter\def\csname foreignlanguage \endcsname{%
2500   \@ifstar\bbl@foreign@s\bbl@foreign@x}
2501 \def\bbl@foreign@x#1#2{%
2502   \begingroup
2503     \let\BabelText\@firstofone
2504     \bbl@beforeforeign
2505     \foreign@language{#1}%
2506     \bbl@usehooks{foreign}{}%
2507     \BabelText{#2}% Now in horizontal mode!
2508   \endgroup}
2509 \def\bbl@foreign@s#1#2{% TODO - \shapemode, \@setpar, ?\@@par
2510   \begingroup
2511     {\par}%
2512     \let\BabelText\@firstofone
2513     \foreign@language{#1}%
2514     \bbl@usehooks{foreign*}{}%
2515     \bbl@dirparastext
2516     \BabelText{#2}% Still in vertical mode!
2517     {\par}%
2518   \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`.

```
2519 \def\foreign@language#1{%
2520   \edef\languagename{#1}%
2521   \bbl@fixname\languagename
2522   \bbl@iflanguage\languagename{%
2523     \expandafter\ifx\csname date\languagename\endcsname\relax
2524       \bbl@warning
```

```
2525        {Unknown language `#1'. Either you have\\%
2526         misspelled its name, it has not been installed,\\%
2527         or you requested it in a previous run. Fix its name,\\%
2528         install it or just rerun the file, respectively.\\%
2529         I'll proceed, but expect wrong results.\\%
2530         Reported}%
2531    \fi
2532    \let\bbl@select@type\@ne
2533    \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.

```
2534 \let\bbl@hyphlist\@empty
2535 \let\bbl@hyphenation@\relax
2536 \let\bbl@pttnlist\@empty
2537 \let\bbl@patterns@\relax
2538 \let\bbl@hymapsel=\@cclv
2539 \def\bbl@patterns#1{%
2540   \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
2541       \csname l@#1\endcsname
2542       \edef\bbl@tempa{#1}%
2543     \else
2544       \csname l@#1:\f@encoding\endcsname
2545       \edef\bbl@tempa{#1:\f@encoding}%
2546     \fi
2547   \@expandtwoargs\bbl@usehooks{patterns}{{#1}{\bbl@tempa}}%
2548   \@ifundefined{bbl@hyphenation@}{}{% Can be \relax!
2549     \begingroup
2550       \bbl@xin@{,\number\language,}{,\bbl@hyphlist}%
2551       \ifin@\else
2552         \@expandtwoargs\bbl@usehooks{hyphenation}{{#1}{\bbl@tempa}}%
2553         \hyphenation{%
2554           \bbl@hyphenation@
2555           \@ifundefined{bbl@hyphenation@#1}%
2556             \@empty
2557             {\space\csname bbl@hyphenation@#1\endcsname}}%
2558         \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
2559       \fi
2560     \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*.

```
2561 \def\hyphenrules#1{%
2562   \edef\bbl@tempf{#1}%
2563   \bbl@fixname\bbl@tempf
2564   \bbl@iflanguage\bbl@tempf{%
2565     \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
2566     \languageshorthands{none}%
```

117

```
2567        \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax
2568          \set@hyphenmins\tw@\thr@@\relax
2569        \else
2570          \expandafter\expandafter\expandafter\set@hyphenmins
2571          \csname\bbl@tempf hyphenmins\endcsname\relax
2572        \fi}}
2573 \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.

```
2574 \def\providehyphenmins#1#2{%
2575   \expandafter\ifx\csname #1hyphenmins\endcsname\relax
2576     \@namedef{#1hyphenmins}{#2}%
2577   \fi}
```

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

```
2578 \def\set@hyphenmins#1#2{%
2579   \lefthyphenmin#1\relax
2580   \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.

```
2581 \ifx\ProvidesFile\@undefined
2582   \def\ProvidesLanguage#1[#2 #3 #4]{%
2583     \wlog{Language: #1 #4 #3 <#2>}%
2584     }
2585 \else
2586   \def\ProvidesLanguage#1{%
2587     \begingroup
2588       \catcode`\ 10 %
2589       \@makeother\/%
2590       \@ifnextchar[%
2591         {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}}
2592   \def\@provideslanguage#1[#2]{%
2593     \wlog{Language: #1 #2}%
2594     \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
2595     \endgroup}
2596 \fi
```

\LdfInit  This macro is defined in two versions. The first version is to be part of the 'kernel' of babel,
ie. the part that is loaded in the format; the second version is defined in babel.def. The
version in the format just checks the category code of the ampersand and then loads
babel.def.
The category code of the ampersand is restored and the macro calls itself again with the
new definition from babel.def

```
2597 \def\LdfInit{%
2598   \chardef\atcatcode=\catcode`\@
2599   \catcode`\@=11\relax
2600   \input babel.def\relax
2601   \catcode`\@=\atcatcode \let\atcatcode\relax
2602   \LdfInit}
```

\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.

```
2603 \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 initialises the save mechanism, \babel@beginsave, is not considered to be
undefined.

```
2604 \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':

```
2605 \providecommand\setlocale{%
2606   \bbl@error
2607     {Not yet available}%
2608     {Find an armchair, sit down and wait}}
2609 \let\uselocale\setlocale
2610 \let\locale\setlocale
2611 \let\selectlocale\setlocale
2612 \let\textlocale\setlocale
2613 \let\textlanguage\setlocale
2614 \let\languagetext\setlocale
```

## 10.2   Errors

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

\@noopterr   When the package was loaded without options not everything will work as expected. An
error message is issued in that case.
When the format knows about \PackageError it must be LaTeX 2$_\varepsilon$, so we can safely use its
error handling interface. Otherwise we'll have to 'keep it simple'.

```
2615 \edef\bbl@nulllanguage{\string\language=0}
2616 \ifx\PackageError\@undefined
2617   \def\bbl@error#1#2{%
2618     \begingroup
2619       \newlinechar=`\^^J
2620       \def\\{^^J(babel) }%
2621       \errhelp{#2}\errmessage{\\#1}%
2622     \endgroup}
2623   \def\bbl@warning#1{%
2624     \begingroup
2625       \newlinechar=`\^^J
2626       \def\\{^^J(babel) }%
2627       \message{\\#1}%
2628     \endgroup}
2629   \def\bbl@info#1{%
2630     \begingroup
2631       \newlinechar=`\^^J
2632       \def\\{^^J}%
2633       \wlog{#1}%
2634     \endgroup}
2635 \else
2636   \def\bbl@error#1#2{%
2637     \begingroup
```

```
2638        \def\\{\MessageBreak}%
2639        \PackageError{babel}{#1}{#2}%
2640      \endgroup}
2641    \def\bbl@warning#1{%
2642      \begingroup
2643        \def\\{\MessageBreak}%
2644        \PackageWarning{babel}{#1}%
2645      \endgroup}
2646    \def\bbl@info#1{%
2647      \begingroup
2648        \def\\{\MessageBreak}%
2649        \PackageInfo{babel}{#1}%
2650      \endgroup}
2651 \fi
2652 \@ifpackagewith{babel}{silent}
2653    {\let\bbl@info\@gobble
2654     \let\bbl@warning\@gobble}
2655    {}
2656 \def\bbl@nocaption#1#2{% 1: text to be printed 2: caption macro \langXname
2657    \gdef#2{\textbf{?#1?}}%
2658    #2%
2659    \bbl@warning{%
2660      \string#2 not set. Please, define\\%
2661      it in the preamble with something like:\\%
2662      \string\renewcommand\string#2{..}\\%
2663      Reported}}
2664 \def\@nolanerr#1{%
2665    \bbl@error
2666      {You haven't defined the language #1\space yet}%
2667      {Your command will be ignored, type <return> to proceed}}
2668 \def\@nopatterns#1{%
2669    \bbl@warning
2670      {No hyphenation patterns were preloaded for\\%
2671       the language `#1' into the format.\\%
2672       Please, configure your TeX system to add them and\\%
2673       rebuild the format. Now I will use the patterns\\%
2674       preloaded for \bbl@nulllanguage\space instead}}
2675 \let\bbl@usehooks\@gobbletwo
2676 ⟨/kernel⟩
2677 ⟨∗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.
toks8 stores info to be shown when the program is run.
We want to add a message to the message LATEX 2.09 puts in the \everyjob register. This
could be done by the following code:

```
\let\orgeveryjob\everyjob
\def\everyjob#1{%
  \orgeveryjob{#1}%
  \orgeveryjob\expandafter{\the\orgeveryjob\immediate\write16{%
      hyphenation patterns for \the\loaded@patterns loaded.}}%
  \let\everyjob\orgeveryjob\let\orgeveryjob\@undefined}
```

The code above redefines the control sequence \everyjob in order to be able to add something to the current contents of the register. This is necessary because the processing of hyphenation patterns happens long before LaTeX fills the register.
There are some problems with this approach though.

- When someone wants to use several hyphenation patterns with SLiTeX the above scheme won't work. The reason is that SLiTeX overwrites the contents of the \everyjob register with its own message.

- Plain TeX does not use the \everyjob register so the message would not be displayed.

To circumvent this a 'dirty trick' can be used. As this code is only processed when creating a new format file there is one command that is sure to be used, \dump. Therefore the original \dump is saved in \org@dump and a new definition is supplied.
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.

```
2678 ⟨⟨Make sure ProvidesFile is defined⟩⟩
2679 \ProvidesFile{hyphen.cfg}[⟨⟨date⟩⟩ ⟨⟨version⟩⟩ Babel hyphens]
2680 \xdef\bbl@format{\jobname}
2681 \ifx\AtBeginDocument\@undefined
2682   \def\@empty{}
2683   \let\orig@dump\dump
2684   \def\dump{%
2685     \ifx\@ztryfc\@undefined
2686     \else
2687       \toks0=\expandafter{\@preamblecmds}%
2688       \edef\@preamblecmds{\noexpand\@begindocumenthook\the\toks0}%
2689       \def\@begindocumenthook{}%
2690     \fi
2691     \let\dump\orig@dump\let\orig@dump\@undefined\dump}
2692 \fi
2693 ⟨⟨Define core switching macros⟩⟩
2694 \toks8{Babel <<@version@>> and hyphenation patterns for }%
```

\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.

```
2695 \def\process@line#1#2 #3 #4 {%
2696   \ifx=#1%
2697     \process@synonym{#2}%
2698   \else
2699     \process@language{#1#2}{#3}{#4}%
2700   \fi
2701   \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.

```
2702 \toks@{}
2703 \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.

```
2704 \def\process@synonym#1{%
2705   \ifnum\last@language=\m@ne
2706     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
2707   \else
2708     \expandafter\chardef\csname l@#1\endcsname\last@language
2709     \wlog{\string\l@#1=\string\language\the\last@language}%
2710     \expandafter\let\csname #1hyphenmins\expandafter\endcsname
2711       \csname\languagename hyphenmins\endcsname
2712     \let\bbl@elt\relax
2713     \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}{}{}}%
2714   \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 'name' of the language that will be loaded now is added to the token register \toks8. and finally 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 languagues 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.

```
2715 \def\process@language#1#2#3{%
2716   \expandafter\addlanguage\csname l@#1\endcsname
2717   \expandafter\language\csname l@#1\endcsname
2718   \edef\languagename{#1}%
2719   \bbl@hook@everylanguage{#1}%
2720   \bbl@get@enc#1::\@@@
```

```
2721   \begingroup
2722     \lefthyphenmin\m@ne
2723     \bbl@hook@loadpatterns{#2}%
2724     \ifnum\lefthyphenmin=\m@ne
2725     \else
2726       \expandafter\xdef\csname #1hyphenmins\endcsname{%
2727         \the\lefthyphenmin\the\righthyphenmin}%
2728     \fi
2729   \endgroup
2730   \def\bbl@tempa{#3}%
2731   \ifx\bbl@tempa\@empty\else
2732     \bbl@hook@loadexceptions{#3}%
2733   \fi
2734   \let\bbl@elt\relax
2735   \edef\bbl@languages{%
2736     \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
2737   \ifnum\the\language=\z@
2738     \expandafter\ifx\csname #1hyphenmins\endcsname\relax
2739       \set@hyphenmins\tw@\thr@@\relax
2740     \else
2741       \expandafter\expandafter\expandafter\set@hyphenmins
2742         \csname #1hyphenmins\endcsname
2743     \fi
2744     \the\toks@
2745     \toks@{}%
2746   \fi}
```

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

```
2747 \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.

```
2748 \def\bbl@hook@everylanguage#1{}
2749 \def\bbl@hook@loadpatterns#1{\input #1\relax}
2750 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
2751 \let\bbl@hook@loadkernel\bbl@hook@loadpatterns
2752 \begingroup
2753   \def\AddBabelHook#1#2{%
2754     \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
2755       \def\next{\toks1}%
2756     \else
2757       \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname####1}%
2758     \fi
2759     \next}
2760   \ifx\directlua\@undefined
2761     \ifx\XeTeXinputencoding\@undefined\else
2762       \input xebabel.def
2763     \fi
2764   \else
2765     \input luababel.def
2766   \fi
2767   \openin1 = babel-\bbl@format.cfg
2768   \ifeof1
2769   \else
2770     \input babel-\bbl@format.cfg\relax
2771   \fi
2772   \closein1
```

```
2773 \endgroup
2774 \bbl@hook@loadkernel{switch.def}
```

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

```
2775 \openin1 = language.dat
```

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

```
2776 \def\languagename{english}%
2777 \ifeof1
2778   \message{I couldn't find the file language.dat,\space
2779           I will try the file hyphen.tex}
2780   \input hyphen.tex\relax
2781   \chardef\l@english\z@
2782 \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.

```
2783   \last@language\m@ne
```

We now read lines from the file until the end is found

```
2784   \loop
```

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.

```
2785     \endlinechar\m@ne
2786     \read1 to \bbl@line
2787     \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.

```
2788     \if T\ifeof1F\fi T\relax
2789       \ifx\bbl@line\@empty\else
2790         \edef\bbl@line{\bbl@line\space\space\space}%
2791         \expandafter\process@line\bbl@line\relax
2792       \fi
2793   \repeat
```

Check for the end of the file. We must reverse the test for \ifeof without \else. Then reactivate the default patterns.

```
2794   \begingroup
2795     \def\bbl@elt#1#2#3#4{%
2796       \global\language=#2\relax
2797       \gdef\languagename{#1}%
2798       \def\bbl@elt##1##2##3##4{}}%
2799     \bbl@languages
2800   \endgroup
2801 \fi
```

and close the configuration file.

```
2802 \closein1
```

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

```
2803 \if/\the\toks@/\else
```

```
2804    \errhelp{language.dat loads no language, only synonyms}
2805    \errmessage{Orphan language synonym}
2806 \fi
2807 \advance\last@language\@ne
2808 \edef\bbl@tempa{%
2809    \everyjob{%
2810       \the\everyjob
2811       \ifx\typeout\@undefined
2812          \immediate\write16%
2813       \else
2814          \noexpand\typeout
2815       \fi
2816       {\the\toks8 \the\last@language\space language(s) loaded.}}}
2817 \advance\last@language\m@ne
2818 \bbl@tempa
```

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.

```
2819 \let\bbl@line\@undefined
2820 \let\process@line\@undefined
2821 \let\process@synonym\@undefined
2822 \let\process@language\@undefined
2823 \let\bbl@get@enc\@undefined
2824 \let\bbl@hyph@enc\@undefined
2825 \let\bbl@tempa\@undefined
2826 \let\bbl@hook@loadkernel\@undefined
2827 \let\bbl@hook@everylanguage\@undefined
2828 \let\bbl@hook@loadpatterns\@undefined
2829 \let\bbl@hook@loadexceptions\@undefined
2830 ⟨/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].

```
2831 ⟨⟨*More package options⟩⟩ ≡
2832 \ifodd\bbl@engine
2833    \DeclareOption{bidi=basic-r}%
2834       {\let\bbl@beforeforeign\leavevmode
2835       \newattribute\bbl@attr@dir
2836       \bbl@exp{\output{\bodydir\pagedir\the\output}}%
2837       \AtEndOfPackage{\EnableBabelHook{babel-bidi}}}
2838    \DeclareOption{bidi=basic}%  TODO -- copypaste pattern
2839       {\let\bbl@beforeforeign\leavevmode
2840       \newattribute\bbl@attr@dir
2841       \bbl@exp{\output{\bodydir\pagedir\the\output}}%
2842       \AtEndOfPackage{\EnableBabelHook{babel-bidi}}}
2843 \else
2844    \DeclareOption{bidi=basic-r}%
2845       {\bbl@error
2846          {The bidi method `basic-r' is available only in\\%
2847          luatex. I'll continue with `bidi=default', so\\%
2848          expect wrong results}%
```

```
2849       {See the manual for further details.}%
2850     \let\bbl@beforeforeign\leavevmode
2851     \AtEndOfPackage{%
2852       \EnableBabelHook{babel-bidi}%
2853       \bbl@xebidipar}}
2854   \DeclareOption{bidi=basic}%    TODO --- copypaste pattern
2855     {\bbl@error
2856       {The bidi method `basic' is available only in\\%
2857        luatex. I'll continue with `bidi=default', so\\%
2858        expect wrong results}%
2859       {See the manual for further details.}%
2860     \let\bbl@beforeforeign\leavevmode
2861     \AtEndOfPackage{%
2862       \EnableBabelHook{babel-bidi}%
2863       \bbl@xebidipar}}
2864 \fi
2865 \DeclareOption{bidi=default}%
2866   {\let\bbl@beforeforeign\leavevmode
2867    \ifodd\bbl@engine
2868      \newattribute\bbl@attr@dir
2869      \bbl@exp{\output{\bodydir\pagedir\the\output}}%
2870    \fi
2871    \AtEndOfPackage{%
2872      \EnableBabelHook{babel-bidi}%
2873      \ifodd\bbl@engine\else
2874        \bbl@xebidipar
2875      \fi}}
2876 ⟨⟨/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.

```
2877 ⟨⟨∗Font selection⟩⟩ ≡
2878 \bbl@trace{Font handling with fontspec}
2879 \@onlypreamble\babelfont
2880 \newcommand\babelfont[2][]{%  1=langs/scripts 2=fam
2881   \edef\bbl@tempa{#1}%
2882   \def\bbl@tempb{#2}%
2883   \ifx\fontspec\@undefined
2884     \usepackage{fontspec}%
2885   \fi
2886   \EnableBabelHook{babel-fontspec}%
2887   \bbl@bblfont}
2888 \newcommand\bbl@bblfont[2][]{% 1=features 2=fontname
2889   \bbl@ifunset{\bbl@tempb family}{\bbl@providefam{\bbl@tempb}}{}%
2890   \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
2891   \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
2892     {\bbl@csarg\edef{\bbl@tempb dflt@}{<>{#1}{#2}}% save bbl@rmdflt@
2893      \bbl@exp{%
2894        \let\<bbl@\bbl@tempb dflt@\languagename>\<bbl@\bbl@tempb dflt@>%
2895        \\\bbl@font@set\<bbl@\bbl@tempb dflt@\languagename>%
2896                      \<\bbl@tempb default>\<\bbl@tempb family>}}%
2897     {\bbl@foreach\bbl@tempa{% ie bbl@rmdflt@lang / *scrt
2898        \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:

```
2899 \def\bbl@providefam#1{%
2900   \bbl@exp{%
2901     \\\newcommand\<#1default>{}% Just define it
2902     \\\bbl@add@list\\\bbl@font@fams{#1}%
```

126

```
2903      \\\DeclareRobustCommand\<#1family>{%
2904        \\\not@math@alphabet\<#1family>\relax
2905        \\\fontfamily\<#1default>\\\selectfont}%
2906      \\\DeclareTextFontCommand{\<text#1>}{\<#1family>}}}
```

The following macro is activated when the hook `babel-fontspec` is enabled.

```
2907 \def\bbl@switchfont{%
2908   \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
2909   \bbl@exp{%  eg Arabic -> arabic
2910     \lowercase{\edef\\\bbl@tempa{\bbl@cs{sname@\languagename}}}}%
2911   \bbl@foreach\bbl@font@fams{%
2912     \bbl@ifunset{bbl@##1dflt@\languagename}%       (1) language?
2913       {\bbl@ifunset{bbl@##1dflt@*\bbl@tempa}%      (2) from script?
2914          {\bbl@ifunset{bbl@##1dflt@}%              2=F - (3) from generic?
2915            {}%                                     123=F - nothing!
2916            {\bbl@exp{%                             3=T - from generic
2917              \global\let\<bbl@##1dflt@\languagename>%
2918                        \<bbl@##1dflt@>}}}%
2919          {\bbl@exp{%                               2=T - from script
2920            \global\let\<bbl@##1dflt@\languagename>%
2921                      \<bbl@##1dflt@*\bbl@tempa>}}}%
2922       {}}%                                         1=T - language, already defined
2923   \def\bbl@tempa{%
2924     \bbl@warning{The current font is not a standard family:\\%
2925       \fontname\font\\%
2926       Script and Language are not applied. Consider defining a\\%
2927       new family with \string\babelfont. Reported}}%
2928   \bbl@foreach\bbl@font@fams{%     don't gather with prev for
2929     \bbl@ifunset{bbl@##1dflt@\languagename}%
2930       {\bbl@cs{famrst@##1}%
2931        \global\bbl@csarg\let{famrst@##1}\relax}%
2932       {\bbl@exp{% order is relevant
2933          \\\bbl@add\\\originalTeX{%
2934            \\\bbl@font@rst{\bbl@cs{##1dflt@\languagename}}%
2935                          \<##1default>\<##1family>{##1}}%
2936          \\\bbl@font@set\<bbl@##1dflt@\languagename>% the main part!
2937                        \<##1default>\<##1family>}}}%
2938   \bbl@ifrestoring{}{\bbl@tempa}}%
```

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.

```
2939 \def\bbl@font@set#1#2#3{% eg \bbl@rmdflt@lang \rmdefault \rmfamily
2940   \bbl@xin@{<>}{#1}%
2941   \ifin@
2942     \bbl@exp{\\\bbl@fontspec@set\\#1\expandafter\@gobbletwo#1}%
2943   \fi
2944   \bbl@exp{%
2945     \def\\#2{#1}%          eg, \rmdefault{\bbl@rmdflt@lang}
2946     \\\bbl@ifsamestring{#2}{\f@family}{\\#3\let\\\bbl@tempa\relax}{}}}
2947 \def\bbl@fontspec@set#1#2#3{% eg \bbl@rmdflt@lang fnt-opt fnt-nme
2948   \let\bbl@tempe\bbl@mapselect
2949   \let\bbl@mapselect\relax
2950   \bbl@exp{\<fontspec_set_family:Nnn>\\#1%
2951     {\bbl@cs{lsys@\languagename},#2}}{#3}%
2952   \let\bbl@mapselect\bbl@tempe
2953   \bbl@toglobal#1}%
```

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.

```
2954 \def\bbl@font@rst#1#2#3#4{%
2955   \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.

```
2956 \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 :-).

```
2957 \newcommand\babelFSstore[2][]{%
2958   \bbl@ifblank{#1}%
2959     {\bbl@csarg\def{sname@#2}{Latin}}%
2960     {\bbl@csarg\def{sname@#2}{#1}}%
2961   \bbl@provide@dirs{#2}%
2962   \bbl@csarg\ifnum{wdir@#2}>\z@
2963     \let\bbl@beforeforeign\leavevmode
2964     \EnableBabelHook{babel-bidi}%
2965   \fi
2966   \bbl@foreach{#2}{%
2967     \bbl@FSstore{##1}{rm}\rmdefault\bbl@save@rmdefault
2968     \bbl@FSstore{##1}{sf}\sfdefault\bbl@save@sfdefault
2969     \bbl@FSstore{##1}{tt}\ttdefault\bbl@save@ttdefault}}
2970 \def\bbl@FSstore#1#2#3#4{%
2971   \bbl@csarg\edef{#2default#1}{#3}%
2972   \expandafter\addto\csname extras#1\endcsname{%
2973     \let#4#3%
2974     \ifx#3\f@family
2975       \edef#3{\csname bbl@#2default#1\endcsname}%
2976       \fontfamily{#3}\selectfont
2977     \else
2978       \edef#3{\csname bbl@#2default#1\endcsname}%
2979     \fi}%
2980   \expandafter\addto\csname noextras#1\endcsname{%
2981     \ifx#3\f@family
2982       \fontfamily{#4}\selectfont
2983     \fi
2984     \let#3#4}}
2985 \let\bbl@langfeatures\@empty
2986 \def\babelFSfeatures{% make sure \fontspec is redefined once
2987   \let\bbl@ori@fontspec\fontspec
2988   \renewcommand\fontspec[1][]{%
2989     \bbl@ori@fontspec[\bbl@langfeatures##1]}
2990   \let\babelFSfeatures\bbl@FSfeatures
2991   \babelFSfeatures}
2992 \def\bbl@FSfeatures#1#2{%
2993   \expandafter\addto\csname extras#1\endcsname{%
2994     \babel@save\bbl@langfeatures
2995     \edef\bbl@langfeatures{#2,}}}
2996 ⟨⟨/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.

LATEX sets many "codes" just before loading hyphen.cfg. That is not a problem in luatex, but in xetex they must be reset to the proper value. Most of the work is done in xe(la)tex.ini, so here we just "undo" some of the changes done by LATEX. Anyway, for consistency LuaTEX also resets the catcodes.

```
2997 ⟨⟨∗Restore Unicode catcodes before loading patterns⟩⟩ ≡
2998  \begingroup
2999      % Reset chars "80-"C0 to category "other", no case mapping:
3000    \catcode`\@=11 \count@=128
3001    \loop\ifnum\count@<192
3002      \global\uccode\count@=0 \global\lccode\count@=0
3003      \global\catcode\count@=12 \global\sfcode\count@=1000
3004      \advance\count@ by 1 \repeat
3005      % Other:
3006    \def\O ##1 {%
3007      \global\uccode"##1=0 \global\lccode"##1=0
3008      \global\catcode"##1=12 \global\sfcode"##1=1000 }%
3009      % Letter:
3010    \def\L ##1 ##2 ##3 {\global\catcode"##1=11
3011      \global\uccode"##1="##2
3012      \global\lccode"##1="##3
3013      % Uppercase letters have sfcode=999:
3014      \ifnum"##1="##3 \else \global\sfcode"##1=999 \fi }%
3015      % Letter without case mappings:
3016    \def\l ##1 {\L ##1 ##1 ##1 }%
3017    \l 00AA
3018    \L 00B5 039C 00B5
3019    \l 00BA
3020    \O 00D7
3021    \l 00DF
3022    \O 00F7
3023    \L 00FF 0178 00FF
3024  \endgroup
3025  \input #1\relax
3026 ⟨⟨/Restore Unicode catcodes before loading patterns⟩⟩
```

Some more common code.

```
3027 ⟨⟨∗Footnote changes⟩⟩ ≡
3028 \bbl@trace{Bidi footnotes}
3029 \ifx\bbl@beforeforeign\leavevmode
3030   \def\bbl@footnote#1#2#3{%
3031     \@ifnextchar[%
3032       {\bbl@footnote@o{#1}{#2}{#3}}%
3033       {\bbl@footnote@x{#1}{#2}{#3}}}
3034   \def\bbl@footnote@x#1#2#3#4{%
3035     \bgroup
3036       \select@language@x{\bbl@main@language}%
3037       \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
3038     \egroup}
3039   \def\bbl@footnote@o#1#2#3[#4]#5{%
3040     \bgroup
3041       \select@language@x{\bbl@main@language}%
3042       \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
3043     \egroup}
3044   \def\bbl@footnotetext#1#2#3{%
3045     \@ifnextchar[%
3046       {\bbl@footnotetext@o{#1}{#2}{#3}}%
3047       {\bbl@footnotetext@x{#1}{#2}{#3}}}
3048   \def\bbl@footnotetext@x#1#2#3#4{%
```

```
3049    \bgroup
3050      \select@language@x{\bbl@main@language}%
3051      \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
3052    \egroup}
3053  \def\bbl@footnotetext@o#1#2#3[#4]#5{%
3054    \bgroup
3055      \select@language@x{\bbl@main@language}%
3056      \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
3057    \egroup}
3058  \def\BabelFootnote#1#2#3#4{%
3059    \ifx\bbl@fn@footnote\@undefined
3060      \let\bbl@fn@footnote\footnote
3061    \fi
3062    \ifx\bbl@fn@footnotetext\@undefined
3063      \let\bbl@fn@footnotetext\footnotetext
3064    \fi
3065    \bbl@ifblank{#2}%
3066      {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
3067       \@namedef{\bbl@stripslash#1text}%
3068         {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
3069      {\def#1{\bbl@exp{\\\bbl@footnote{\\\foreignlanguage{#2}}}{#3}{#4}}%
3070       \@namedef{\bbl@stripslash#1text}%
3071         {\bbl@exp{\\\bbl@footnotetext{\\\foreignlanguage{#2}}}{#3}{#4}}}}
3072 \fi
3073 ⟨⟨/Footnote changes⟩⟩
```

Now, the code.

```
3074 ⟨∗xetex⟩
3075 \def\BabelStringsDefault{unicode}
3076 \let\xebbl@stop\relax
3077 \AddBabelHook{xetex}{encodedcommands}{%
3078   \def\bbl@tempa{#1}%
3079   \ifx\bbl@tempa\@empty
3080     \XeTeXinputencoding"bytes"%
3081   \else
3082     \XeTeXinputencoding"#1"%
3083   \fi
3084   \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
3085 \AddBabelHook{xetex}{stopcommands}{%
3086   \xebbl@stop
3087   \let\xebbl@stop\relax}
3088 \AddBabelHook{xetex}{loadkernel}{%
3089 ⟨⟨Restore Unicode catcodes before loading patterns⟩⟩}
3090 \ifx\DisableBabelHook\@undefined\endinput\fi
3091 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
3092 \DisableBabelHook{babel-fontspec}
3093 ⟨⟨Font selection⟩⟩
3094 \input txtbabel.def
3095 ⟨/xetex⟩
```

## 13.2  Layout

*In progress.*
Unfortunately, proper support for xetex requires patching somehow lots of macros and packages (and some issues related to \specials remain, like color and hyperlinks). At least at this stage, babel will not do it and therefore a package like bidi (by Vafa Khalighi) would be necessary to overcome the limitations of xetex. Any help in making babel and bidi collaborate will be welcome, although the underlying concepts in both packages seem very

different. Note also 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.

```
3096 ⟨∗texxet⟩
3097 \bbl@trace{Redefinitions for bidi layout}
3098 \def\bbl@sspre@caption{%
3099   \bbl@exp{\everyhbox{\\\bbl@textdir\bbl@cs{wdir@\bbl@main@language}}}}}
3100 \ifx\bbl@opt@layout\@nnil\endinput\fi  % No layout
3101 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
3102 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
3103 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
3104   \def\@hangfrom#1{%
3105     \setbox\@tempboxa\hbox{{#1}}%
3106     \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
3107     \noindent\box\@tempboxa}
3108   \def\raggedright{%
3109     \let\\\@centercr
3110     \bbl@startskip\z@skip
3111     \@rightskip\@flushglue
3112     \bbl@endskip\@rightskip
3113     \parindent\z@
3114     \parfillskip\bbl@startskip}
3115   \def\raggedleft{%
3116     \let\\\@centercr
3117     \bbl@startskip\@flushglue
3118     \bbl@endskip\z@skip
3119     \parindent\z@
3120     \parfillskip\bbl@endskip}
3121 \fi
3122 \IfBabelLayout{lists}
3123   {\def\list#1#2{%
3124     \ifnum \@listdepth >5\relax
3125       \@toodeep
3126     \else
3127       \global\advance\@listdepth\@ne
3128     \fi
3129     \rightmargin\z@
3130     \listparindent\z@
3131     \itemindent\z@
3132     \csname @list\romannumeral\the\@listdepth\endcsname
3133     \def\@itemlabel{#1}%
3134     \let\makelabel\@mklab
3135     \@nmbrlistfalse
3136     #2\relax
3137     \@trivlist
3138     \parskip\parsep
3139     \parindent\listparindent
3140     \advance\linewidth-\rightmargin
3141     \advance\linewidth-\leftmargin
3142     \advance\@totalleftmargin
3143       \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi
3144     \parshape\@ne\@totalleftmargin\linewidth
3145     \ignorespaces}%
3146   \ifcase\bbl@engine
```

```
3147      \def\labelenumii{)\theenumii(}%
3148      \def\p@enumiii{\p@enumii)\theenumii(}%
3149    \fi
3150    \def\@verbatim{%
3151      \trivlist \item\relax
3152      \if@minipage\else\vskip\parskip\fi
3153      \bbl@startskip\textwidth
3154      \advance\bbl@startskip-\linewidth
3155      \bbl@endskip\z@skip
3156      \parindent\z@
3157      \parfillskip\@flushglue
3158      \parskip\z@skip
3159      \@@par
3160      \language\l@nohyphenation
3161      \@tempswafalse
3162      \def\par{%
3163        \if@tempswa
3164          \leavevmode\null
3165          \@@par\penalty\interlinepenalty
3166        \else
3167          \@tempswatrue
3168          \ifhmode\@@par\penalty\interlinepenalty\fi
3169        \fi}%
3170      \let\do\@makeother \dospecials
3171      \obeylines \verbatim@font \@noligs
3172      \everypar\expandafter{\the\everypar\unpenalty}}}
3173    {}
3174 \IfBabelLayout{contents}
3175    {\def\@dottedtocline#1#2#3#4#5{%
3176        \ifnum#1>\c@tocdepth\else
3177          \vskip \z@ \@plus.2\p@
3178          {\bbl@startskip#2\relax
3179           \bbl@endskip\@tocrmarg
3180           \parfillskip-\bbl@endskip
3181           \parindent#2\relax
3182           \@afterindenttrue
3183           \interlinepenalty\@M
3184           \leavevmode
3185           \@tempdima#3\relax
3186           \advance\bbl@startskip\@tempdima
3187           \null\nobreak\hskip-\bbl@startskip
3188           {#4}\nobreak
3189           \leaders\hbox{%
3190             $\m@th\mkern\@dotsep mu\hbox{.}\mkern\@dotsep mu$}%
3191           \hfill\nobreak
3192           \hb@xt@\@pnumwidth{\hfil\normalfont\normalcolor#5}%
3193           \par}%
3194        \fi}}
3195    {}
3196 \IfBabelLayout{columns}
3197    {\def\@outputdblcol{%
3198        \if@firstcolumn
3199          \global\@firstcolumnfalse
3200          \global\setbox\@leftcolumn\copy\@outputbox
3201          \splitmaxdepth\maxdimen
3202          \vbadness\maxdimen
3203          \setbox\@outputbox\vbox{\unvbox\@outputbox\unskip}%
3204          \setbox\@outputbox\vsplit\@outputbox to\maxdimen
3205          \toks@\expandafter{\topmark}%
```

```
3206        \xdef\@firstcoltopmark{\the\toks@}%
3207        \toks@\expandafter{\splitfirstmark}%
3208        \xdef\@firstcolfirstmark{\the\toks@}%
3209        \ifx\@firstcolfirstmark\@empty
3210          \global\let\@setmarks\relax
3211        \else
3212          \gdef\@setmarks{%
3213            \let\firstmark\@firstcolfirstmark
3214            \let\topmark\@firstcoltopmark}%
3215        \fi
3216      \else
3217        \global\@firstcolumntrue
3218        \setbox\@outputbox\vbox{%
3219          \hb@xt@\textwidth{%
3220            \hskip\columnwidth
3221            \hfil
3222            {\normalcolor\vrule \@width\columnseprule}%
3223            \hfil
3224            \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
3225            \hskip-\textwidth
3226            \hb@xt@\columnwidth{\box\@outputbox \hss}%
3227            \hskip\columnsep
3228            \hskip\columnwidth}}%
3229        \@combinedblfloats
3230        \@setmarks
3231        \@outputpage
3232        \begingroup
3233          \@dblfloatplacement
3234          \@startdblcolumn
3235          \@whilesw\if@fcolmade \fi{\@outputpage
3236          \@startdblcolumn}%
3237        \endgroup
3238      \fi}}%
3239  {}
3240 ⟨⟨Footnote changes⟩⟩
3241 \IfBabelLayout{footnotes}%
3242   {\BabelFootnote\footnote\languagename{}{}%
3243    \BabelFootnote\localfootnote\languagename{}{}%
3244    \BabelFootnote\mainfootnote{}{}{}}
3245   {}
```

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.

```
3246 \IfBabelLayout{counters}%
3247   {\let\bbl@latinarabic=\@arabic
3248    \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
3249    \let\bbl@asciiroman=\@roman
3250    \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciiroman#1}}}%
3251    \let\bbl@asciiRoman=\@Roman
3252    \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}}{}
3253 ⟨/texxet⟩
```

### 13.3   LuaTeX

The new 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 has 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). 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.

```
3254 ⟨*luatex⟩
3255 \ifx\AddBabelHook\@undefined
3256 \bbl@trace{Read language.dat}
3257 \begingroup
3258   \toks@{}
3259   \count@\z@ % 0=start, 1=0th, 2=normal
3260   \def\bbl@process@line#1#2 #3 #4 {%
3261     \ifx=#1%
3262       \bbl@process@synonym{#2}%
3263     \else
3264       \bbl@process@language{#1#2}{#3}{#4}%
3265     \fi
3266     \ignorespaces}
3267   \def\bbl@manylang{%
3268     \ifnum\bbl@last>\@ne
3269       \bbl@info{Non-standard hyphenation setup}%
3270     \fi
3271     \let\bbl@manylang\relax}
3272   \def\bbl@process@language#1#2#3{%
3273     \ifcase\count@
3274       \@ifundefined{zth@#1}{\count@\tw@}{\count@\@ne}%
3275     \or
3276       \count@\tw@
3277     \fi
3278     \ifnum\count@=\tw@
3279       \expandafter\addlanguage\csname l@#1\endcsname
3280       \language\allocationnumber
3281       \chardef\bbl@last\allocationnumber
3282       \bbl@manylang
3283       \let\bbl@elt\relax
```

```
3284        \xdef\bbl@languages{%
3285          \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
3286      \fi
3287      \the\toks@
3288      \toks@{}}
3289    \def\bbl@process@synonym@aux#1#2{%
3290      \global\expandafter\chardef\csname l@#1\endcsname#2\relax
3291      \let\bbl@elt\relax
3292      \xdef\bbl@languages{%
3293        \bbl@languages\bbl@elt{#1}{#2}{}{}}}%
3294    \def\bbl@process@synonym#1{%
3295      \ifcase\count@
3296        \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
3297      \or
3298        \@ifundefined{zth@#1}{\bbl@process@synonym@aux{#1}{0}}{}%
3299      \else
3300        \bbl@process@synonym@aux{#1}{\the\bbl@last}%
3301      \fi}
3302    \ifx\bbl@languages\@undefined % Just a (sensible?) guess
3303      \chardef\l@english\z@
3304      \chardef\l@USenglish\z@
3305      \chardef\bbl@last\z@
3306      \global\@namedef{bbl@hyphendata@0}{{hyphen.tex}{}}
3307      \gdef\bbl@languages{%
3308        \bbl@elt{english}{0}{hyphen.tex}{}%
3309        \bbl@elt{USenglish}{0}{}{}}
3310    \else
3311      \global\let\bbl@languages@format\bbl@languages
3312      \def\bbl@elt#1#2#3#4{% Remove all except language 0
3313        \ifnum#2>\z@\else
3314          \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
3315        \fi}%
3316      \xdef\bbl@languages{\bbl@languages}%
3317    \fi
3318    \def\bbl@elt#1#2#3#4{\@namedef{zth@#1}{}} % Define flags
3319    \bbl@languages
3320    \openin1=language.dat
3321    \ifeof1
3322      \bbl@warning{I couldn't find language.dat. No additional\\%
3323                    patterns loaded. Reported}%
3324    \else
3325      \loop
3326        \endlinechar\m@ne
3327        \read1 to \bbl@line
3328        \endlinechar`\^^M
3329        \if T\ifeof1F\fi T\relax
3330          \ifx\bbl@line\@empty\else
3331            \edef\bbl@line{\bbl@line\space\space\space}%
3332            \expandafter\bbl@process@line\bbl@line\relax
3333          \fi
3334      \repeat
3335    \fi
3336 \endgroup
3337 \bbl@trace{Macros for reading patterns files}
3338 \def\bbl@get@enc#1:#2:#3\@@@{\def\bbl@hyph@enc{#2}}
3339 \ifx\babelcatcodetablenum\@undefined
3340    \def\babelcatcodetablenum{5211}
3341 \fi
3342 \def\bbl@luapatterns#1#2{%
```

135

```
3343    \bbl@get@enc#1::\@@@
3344    \setbox\z@\hbox\bgroup
3345      \begingroup
3346        \ifx\catcodetable\@undefined
3347          \let\savecatcodetable\luatexsavecatcodetable
3348          \let\initcatcodetable\luatexinitcatcodetable
3349          \let\catcodetable\luatexcatcodetable
3350        \fi
3351        \savecatcodetable\babelcatcodetablenum\relax
3352        \initcatcodetable\numexpr\babelcatcodetablenum+1\relax
3353        \catcodetable\numexpr\babelcatcodetablenum+1\relax
3354        \catcode`\#=6  \catcode`\$=3 \catcode`\&=4 \catcode`\^=7
3355        \catcode`\_=8  \catcode`\{=1 \catcode`\}=2 \catcode`\~=13
3356        \catcode`\@=11 \catcode`\^^I=10 \catcode`\^^J=12
3357        \catcode`\<=12 \catcode`\>=12 \catcode`\*=12 \catcode`\.=12
3358        \catcode`\-=12 \catcode`\/=12 \catcode`\[=12 \catcode`\]=12
3359        \catcode`\`=12 \catcode`\'=12 \catcode`\"=12
3360        \input #1\relax
3361        \catcodetable\babelcatcodetablenum\relax
3362      \endgroup
3363      \def\bbl@tempa{#2}%
3364      \ifx\bbl@tempa\@empty\else
3365        \input #2\relax
3366      \fi
3367    \egroup}%
3368 \def\bbl@patterns@lua#1{%
3369    \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
3370      \csname l@#1\endcsname
3371      \edef\bbl@tempa{#1}%
3372    \else
3373      \csname l@#1:\f@encoding\endcsname
3374      \edef\bbl@tempa{#1:\f@encoding}%
3375    \fi\relax
3376    \@namedef{lu@texhyphen@loaded@\the\language}{}% Temp
3377    \@ifundefined{bbl@hyphendata@\the\language}%
3378      {\def\bbl@elt##1##2##3##4{%
3379         \ifnum##2=\csname l@\bbl@tempa\endcsname % #2=spanish, dutch:OT1...
3380           \def\bbl@tempb{##3}%
3381           \ifx\bbl@tempb\@empty\else % if not a synonymous
3382             \def\bbl@tempc{{##3}{##4}}%
3383           \fi
3384           \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
3385         \fi}%
3386       \bbl@languages
3387       \@ifundefined{bbl@hyphendata@\the\language}%
3388         {\bbl@info{No hyphenation patterns were set for\\%
3389                    language '\bbl@tempa'. Reported}}%
3390         {\expandafter\expandafter\expandafter\bbl@luapatterns
3391           \csname bbl@hyphendata@\the\language\endcsname}}{}}
3392 \endinput\fi
3393 \begingroup
3394 \catcode`\%=12
3395 \catcode`\'=12
3396 \catcode`\"=12
3397 \catcode`\:=12
3398 \directlua{
3399   Babel = Babel or {}
3400   function Babel.bytes(line)
3401     return line:gsub("(.)",
```

136

```
3402      function (chr) return unicode.utf8.char(string.byte(chr)) end)
3403    end
3404    function Babel.begin_process_input()
3405      if luatexbase and luatexbase.add_to_callback then
3406        luatexbase.add_to_callback('process_input_buffer',
3407                                   Babel.bytes,'Babel.bytes')
3408      else
3409        Babel.callback = callback.find('process_input_buffer')
3410        callback.register('process_input_buffer',Babel.bytes)
3411      end
3412    end
3413    function Babel.end_process_input ()
3414      if luatexbase and luatexbase.remove_from_callback then
3415        luatexbase.remove_from_callback('process_input_buffer','Babel.bytes')
3416      else
3417        callback.register('process_input_buffer',Babel.callback)
3418      end
3419    end
3420    function Babel.addpatterns(pp, lg)
3421      local lg = lang.new(lg)
3422      local pats = lang.patterns(lg) or ''
3423      lang.clear_patterns(lg)
3424      for p in pp:gmatch('[^%s]+') do
3425        ss = ''
3426        for i in string.utfcharacters(p:gsub('%d', '')) do
3427          ss = ss .. '%d?' .. i
3428        end
3429        ss = ss:gsub('^%%d%?%.', '%%.') .. '%d?'
3430        ss = ss:gsub('%.%%d%?$', '%%.')
3431        pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
3432        if n == 0 then
3433          tex.sprint(
3434            [[\string\csname\space bbl@info\endcsname{New pattern: ]]
3435            .. p .. [[}]])
3436          pats = pats .. ' ' .. p
3437        else
3438          tex.sprint(
3439            [[\string\csname\space bbl@info\endcsname{Renew pattern: ]]
3440            .. p .. [[}]])
3441        end
3442      end
3443      lang.patterns(lg, pats)
3444    end
3445  }
3446  \endgroup
3447  \def\BabelStringsDefault{unicode}
3448  \let\luabbl@stop\relax
3449  \AddBabelHook{luatex}{encodedcommands}{%
3450    \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
3451    \ifx\bbl@tempa\bbl@tempb\else
3452      \directlua{Babel.begin_process_input()}%
3453      \def\luabbl@stop{%
3454        \directlua{Babel.end_process_input()}}%
3455    \fi}%
3456  \AddBabelHook{luatex}{stopcommands}{%
3457    \luabbl@stop
3458    \let\luabbl@stop\relax}
3459  \AddBabelHook{luatex}{patterns}{%
3460    \@ifundefined{bbl@hyphendata@\the\language}%
```

```
3461     {\def\bbl@elt##1##2##3##4{%
3462       \ifnum##2=\csname l@##2\endcsname % #2=spanish, dutch:OT1...
3463         \def\bbl@tempb{##3}%
3464         \ifx\bbl@tempb\@empty\else % if not a synonymous
3465           \def\bbl@tempc{{##3}{##4}}%
3466         \fi
3467         \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
3468       \fi}%
3469     \bbl@languages
3470     \@ifundefined{bbl@hyphendata@\the\language}%
3471       {\bbl@info{No hyphenation patterns were set for\\%
3472                 language '#2'. Reported}}%
3473       {\expandafter\expandafter\expandafter\bbl@luapatterns
3474         \csname bbl@hyphendata@\the\language\endcsname}}{}%
3475   \@ifundefined{bbl@patterns@}{}{%
3476     \begingroup
3477       \bbl@xin@{,\number\language,}{,\bbl@pttnlist}%
3478       \ifin@\else
3479         \ifx\bbl@patterns@\@empty\else
3480           \directlua{ Babel.addpatterns(
3481             [[\bbl@patterns@]], \number\language) }%
3482         \fi
3483         \@ifundefined{bbl@patterns@#1}%
3484           \@empty
3485           {\directlua{ Babel.addpatterns(
3486                 [[\space\csname bbl@patterns@#1\endcsname]],
3487                 \number\language) }}%
3488         \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
3489       \fi
3490     \endgroup}}
3491 \AddBabelHook{luatex}{everylanguage}{%
3492   \def\process@language##1##2##3{%
3493     \def\process@line####1####2 ####3 ####4 {}}}
3494 \AddBabelHook{luatex}{loadpatterns}{%
3495   \input #1\relax
3496   \expandafter\gdef\csname bbl@hyphendata@\the\language\endcsname
3497     {{#1}{}}}
3498 \AddBabelHook{luatex}{loadexceptions}{%
3499   \input #1\relax
3500   \def\bbl@tempb##1##2{{##1}{#1}}%
3501   \expandafter\xdef\csname bbl@hyphendata@\the\language\endcsname
3502     {\expandafter\expandafter\expandafter\bbl@tempb
3503      \csname bbl@hyphendata@\the\language\endcsname}}
```

\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.

```
3504 \@onlypreamble\babelpatterns
3505 \AtEndOfPackage{%
3506   \newcommand\babelpatterns[2][\@empty]{%
3507     \ifx\bbl@patterns@\relax
3508       \let\bbl@patterns@\@empty
3509     \fi
3510     \ifx\bbl@pttnlist\@empty\else
3511       \bbl@warning{%
3512         You must not intermingle \string\selectlanguage\space and\\%
3513         \string\babelpatterns\space or some patterns will not\\%
3514         be taken into account. Reported}%
```

```
3515    \fi
3516    \ifx\@empty#1%
3517      \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
3518    \else
3519      \edef\bbl@tempb{\zap@space#1 \@empty}%
3520      \bbl@for\bbl@tempa\bbl@tempb{%
3521        \bbl@fixname\bbl@tempa
3522        \bbl@iflanguage\bbl@tempa{%
3523          \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
3524            \@ifundefined{bbl@patterns@\bbl@tempa}%
3525              \@empty
3526              {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
3527          #2}}}%
3528    \fi}}
```

Common stuff.

```
3529 \AddBabelHook{luatex}{loadkernel}{%
3530 ⟨⟨Restore Unicode catcodes before loading patterns⟩⟩}
3531 \ifx\DisableBabelHook\@undefined\endinput\fi
3532 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
3533 \DisableBabelHook{babel-fontspec}
3534 ⟨⟨Font selection⟩⟩
```

## 13.4   Layout

**Work in progress**.

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.) and with bidi=basic-r, 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.

```
3535 \bbl@trace{Redefinitions for bidi layout}
3536 \ifx\bbl@opt@layout\@nnil\endinput\fi  % if no layout
3537 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
3538   \def\bbl@nextfake#1{%
3539     \mathdir\bodydir  % non-local, use always inside a group!
3540     \bbl@exp{%
3541       #1%                Once entered in math, set boxes to restore values
3542       \everyvbox{%
3543         \the\everyvbox
3544         \bodydir\the\bodydir
3545         \mathdir\the\mathdir
3546         \everyhbox{\the\everyhbox}%
3547         \everyvbox{\the\everyvbox}}%
3548       \everyhbox{%
3549         \the\everyhbox
3550         \bodydir\the\bodydir
3551         \mathdir\the\mathdir
3552         \everyhbox{\the\everyhbox}%
3553         \everyvbox{\the\everyvbox}}}}%
3554   \def\@hangfrom#1{%
3555     \setbox\@tempboxa\hbox{{#1}}%
3556     \hangindent\wd\@tempboxa
3557     \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
```

```
3558        \shapemode\@ne
3559      \fi
3560      \noindent\box\@tempboxa}
3561 \fi
3562 \IfBabelLayout{tabular}
3563   {\def\@tabular{%
3564      \leavevmode\hbox\bgroup\bbl@nextfake$%    %$
3565      \let\@acol\@tabacol        \let\@classz\@tabclassz
3566      \let\@classiv\@tabclassiv \let\\\@tabularcr\@tabarray}}
3567   {}
3568 \IfBabelLayout{lists}
3569   {\def\list#1#2{%
3570      \ifnum \@listdepth >5\relax
3571        \@toodeep
3572      \else
3573        \global\advance\@listdepth\@ne
3574      \fi
3575      \rightmargin\z@
3576      \listparindent\z@
3577      \itemindent\z@
3578      \csname @list\romannumeral\the\@listdepth\endcsname
3579      \def\@itemlabel{#1}%
3580      \let\makelabel\@mklab
3581      \@nmbrlistfalse
3582      #2\relax
3583      \@trivlist
3584      \parskip\parsep
3585      \parindent\listparindent
3586      \advance\linewidth -\rightmargin
3587      \advance\linewidth -\leftmargin
3588      \advance\@totalleftmargin \leftmargin
3589      \parshape \@ne
3590      \@totalleftmargin \linewidth
3591      \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
3592        \shapemode\tw@
3593      \fi
3594      \ignorespaces}}
3595   {}
```

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-r, but there are some additional readjustments for bidi=default.

```
3596 \IfBabelLayout{counters}%
3597   {\def\@textsuperscript#1{{% lua has separate settings for math
3598      \m@th
3599      \mathdir\pagedir % required with basic-r; ok with default, too
3600      \ensuremath {^{\mbox {\fontsize \sf@size \z@ #1}}}}}%
3601    \let\bbl@latinarabic=\@arabic
3602    \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
3603    \@ifpackagewith{babel}{bidi=default}%
3604      {\let\bbl@asciiroman=\@roman
3605       \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciiroman#1}}}%
3606       \let\bbl@asciiRoman=\@Roman
3607       \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}%
3608       \def\labelenumii(){\theenumii(}%
3609       \def\p@enumiii{\p@enumii)\theenumii(}}{}}{}
3610 ⟨⟨Footnote changes⟩⟩
3611 \IfBabelLayout{footnotes}%
3612   {\BabelFootnote\footnote\languagename{}{}%
```

140

```
3613    \BabelFootnote\localfootnote\languagename{}{}%
3614    \BabelFootnote\mainfootnote{}{}{}}
3615  {}
3616 ⟨/luatex⟩
```

## 13.5  Auto bidi with `basic-r`

The file babel-bidi.lua currently only contains data. It is a large and boring file and it's not shown here. See the generated file.
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.
TODO: math mode (as weak L?)

```
3617 ⟨∗basic-r⟩
3618 Babel = Babel or {}
3619
3620 require('babel-bidi.lua')
3621
3622 local characters = Babel.characters
3623 local ranges = Babel.ranges
3624
3625 local DIR = node.id("dir")
3626
3627 local function dir_mark(head, from, to, outer)
3628   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
3629   local d = node.new(DIR)
3630   d.dir = '+' .. dir
3631   node.insert_before(head, from, d)
3632   d = node.new(DIR)
3633   d.dir = '-' .. dir
3634   node.insert_after(head, to, d)
3635 end
```

141

```
3636
3637 function Babel.pre_otfload_v(head)
3638   -- head = Babel.numbers(head)
3639   head = Babel.bidi(head, true)
3640   return head
3641 end
3642
3643 function Babel.pre_otfload_h(head)
3644   -- head = Babel.numbers(head)
3645   head = Babel.bidi(head, false)
3646   return head
3647 end
3648
3649 function Babel.bidi(head, ispar)
3650   local first_n, last_n        -- first and last char with nums
3651   local last_es                -- an auxiliary 'last' used with nums
3652   local first_d, last_d        -- first and last char in L/R block
3653   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):

```
3654   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
3655   local strong_lr = (strong == 'l') and 'l' or 'r'
3656   local outer = strong
3657
3658   local new_dir = false
3659   local first_dir = false
3660
3661   local last_lr
3662
3663   local type_n = ''
3664
3665   for item in node.traverse(head) do
3666
3667     -- three cases: glyph, dir, otherwise
3668     if item.id == node.id'glyph' then
3669
3670       local chardata = characters[item.char]
3671       dir = chardata and chardata.d or nil
3672       if not dir then
3673         for nn, et in ipairs(ranges) do
3674           if item.char < et[1] then
3675             break
3676           elseif item.char <= et[2] then
3677             dir = et[3]
3678             break
3679           end
3680         end
3681       end
3682       dir = dir or 'l'
```

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.

```
3683       if new_dir then
3684         attr_dir = 0
3685         for at in node.traverse(item.attr) do
```

```
3686          if at.number == luatexbase.registernumber'bbl@attr@dir' then
3687            attr_dir = at.value
3688          end
3689        end
3690        if attr_dir == 1 then
3691          strong = 'r'
3692        elseif attr_dir == 2 then
3693          strong = 'al'
3694        else
3695          strong = 'l'
3696        end
3697        strong_lr = (strong == 'l') and 'l' or 'r'
3698        outer = strong_lr
3699        new_dir = false
3700      end
3701
3702      if dir == 'nsm' then dir = strong end            -- W1
```

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

```
3703      dir_real = dir              -- We need dir_real to set strong below
3704      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:

```
3705      if strong == 'al' then
3706        if dir == 'en' then dir = 'an' end                -- W2
3707        if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
3708        strong_lr = 'r'                                   -- W3
3709      end
```

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

```
3710      elseif item.id == node.id'dir' then
3711        new_dir = true
3712        dir = nil
3713      else
3714        dir = nil          -- Not a char
3715      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>.

```
3716      if dir == 'en' or dir == 'an' or dir == 'et' then
3717        if dir ~= 'et' then
3718          type_n = dir
3719        end
3720        first_n = first_n or item
3721        last_n = last_es or item
3722        last_es = nil
3723      elseif dir == 'es' and last_n then -- W3+W6
3724        last_es = item
3725      elseif dir == 'cs' then             -- it's right - do nothing
3726      elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
3727        if strong_lr == 'r' and type_n ~= '' then
3728          dir_mark(head, first_n, last_n, 'r')
```

```
3729      elseif strong_lr == 'l' and first_d and type_n == 'an' then
3730        dir_mark(head, first_n, last_n, 'r')
3731        dir_mark(head, first_d, last_d, outer)
3732        first_d, last_d = nil, nil
3733      elseif strong_lr == 'l' and type_n ~= '' then
3734        last_d = last_n
3735      end
3736      type_n = ''
3737      first_n, last_n = nil, nil
3738    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:

```
3739    if dir == 'l' or dir == 'r' then
3740      if dir ~= outer then
3741        first_d = first_d or item
3742        last_d = item
3743      elseif first_d and dir ~= strong_lr then
3744        dir_mark(head, first_d, last_d, outer)
3745        first_d, last_d = nil, nil
3746      end
3747    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.

```
3748    if dir and not last_lr and dir ~= 'l' and outer == 'r' then
3749      item.char = characters[item.char] and
3750                  characters[item.char].m or item.char
3751    elseif (dir or new_dir) and last_lr ~= item then
3752      local mir = outer .. strong_lr .. (dir or outer)
3753      if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
3754        for ch in node.traverse(node.next(last_lr)) do
3755          if ch == item then break end
3756          if ch.id == node.id'glyph' then
3757            ch.char = characters[ch.char].m or ch.char
3758          end
3759        end
3760      end
3761    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`).

```
3762    if dir == 'l' or dir == 'r' then
3763      last_lr = item
3764      strong = dir_real            -- Don't search back - best save now
3765      strong_lr = (strong == 'l') and 'l' or 'r'
3766    elseif new_dir then
3767      last_lr = nil
3768    end
3769  end
```

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

```
3770  if last_lr and outer == 'r' then
```

```
3771    for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
3772      ch.char = characters[ch.char].m or ch.char
3773    end
3774  end
3775  if first_n then
3776    dir_mark(head, first_n, last_n, outer)
3777  end
3778  if first_d then
3779    dir_mark(head, first_d, last_d, outer)
3780  end
```

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

```
3781  return node.prev(head) or head
3782 end
3783 ⟨/basic-r⟩
```

And here the Lua code for bidi=basic:

```
3784 ⟨∗basic⟩
3785 Babel = Babel or {}
3786
3787 Babel.fontmap = Babel.fontmap or {}
3788 Babel.fontmap[0] = {}        -- l
3789 Babel.fontmap[1] = {}        -- r
3790 Babel.fontmap[2] = {}        -- al/an
3791
3792 function Babel.pre_otfload_v(head)
3793   -- head = Babel.numbers(head)
3794   head = Babel.bidi(head, true)
3795   return head
3796 end
3797
3798 function Babel.pre_otfload_h(head)
3799   -- head = Babel.numbers(head)
3800   head = Babel.bidi(head, false)
3801   return head
3802 end
3803
3804 require('babel-bidi.lua')
3805
3806 local characters = Babel.characters
3807 local ranges = Babel.ranges
3808
3809 local DIR = node.id('dir')
3810 local GLYPH = node.id('glyph')
3811
3812 local function insert_implicit(head, state, outer)
3813   local new_state = state
3814   if state.sim and state.eim and state.sim ~= state.eim then
3815     dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
3816     local d = node.new(DIR)
3817     d.dir = '+' .. dir
3818     node.insert_before(head, state.sim, d)
3819     local d = node.new(DIR)
3820     d.dir = '-' .. dir
3821     node.insert_after(head, state.eim, d)
3822   end
3823   new_state.sim, new_state.eim = nil, nil
3824   return head, new_state
```

```
3825 end
3826
3827 local function insert_numeric(head, state)
3828   local new
3829   local new_state = state
3830   if state.san and state.ean and state.san ~= state.ean then
3831     local d = node.new(DIR)
3832     d.dir = '+TLT'
3833     _, new = node.insert_before(head, state.san, d)
3834     if state.san == state.sim then state.sim = new end
3835     local d = node.new(DIR)
3836     d.dir = '-TLT'
3837     _, new = node.insert_after(head, state.ean, d)
3838     if state.ean == state.eim then state.eim = new end
3839   end
3840   new_state.san, new_state.ean = nil, nil
3841   return head, new_state
3842 end
3843
3844 function Babel.bidi(head, ispar)
3845   local d    -- d is used mainly for computations in a loop
3846   local prev_d = ''
3847   local new_d = false
3848
3849   local nodes = {}
3850   local outer_first = nil
3851
3852   local has_en = false
3853   local first_et = nil
3854
3855   local save_outer
3856   local temp = node.get_attribute(head, luatexbase.registernumber'bbl@attr@dir')
3857   if temp then
3858     temp = temp % 3
3859     save_outer = (temp == 0 and 'l') or
3860                  (temp == 1 and 'r') or
3861                  (temp == 2 and 'al')
3862   elseif ispar then
3863     save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
3864   else
3865     save_outer = ('TRT' == tex.textdir) and 'r' or 'l'
3866   end
3867   local outer = save_outer
3868   local last = outer
3869   -- 'al' is only taken into account in the first, current loop
3870   if save_outer == 'al' then save_outer = 'r' end
3871
3872   local fontmap = Babel.fontmap
3873
3874   for item in node.traverse(head) do
3875
3876     -- three cases: glyph, dir, otherwise
3877     if item.id == GLYPH then
3878
3879       local chardata = characters[item.char]
3880       d = chardata and chardata.d or nil
3881       if not d then
3882         for nn, et in ipairs(ranges) do
3883           if item.char < et[1] then
```

```
3884            break
3885          elseif item.char <= et[2] then
3886            d = et[3]
3887            break
3888          end
3889        end
3890      end
3891      d = d or 'l'
3892
3893      local temp = (d == 'l' and 0) or
3894                   (d == 'r' and 1) or
3895                   (d == 'al' and 2) or
3896                   (d == 'an' and 2) or nil
3897      if temp and fontmap and fontmap[temp][item.font] then
3898        item.font = fontmap[temp][item.font]
3899      end
3900
3901      if new_d then
3902        table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
3903        attr_d = node.get_attribute(item,
3904                     luatexbase.registernumber'bbl@attr@dir')
3905        attr_d = attr_d % 3
3906        if attr_d == 1 then
3907          outer_first = 'r'
3908          last = 'r'
3909        elseif attr_d == 2 then
3910          outer_first = 'r'
3911          last = 'al'
3912        else
3913          outer_first = 'l'
3914          last = 'l'
3915        end
3916        outer = last
3917        has_en = false
3918        first_et = nil
3919        new_d = false
3920      end
3921
3922      -- Optimizar el caso en que el dir actual == anterior
3923      -- eg, if prev_dir == d goto next_node end
3924
3925    elseif item.id == DIR then
3926      d = nil
3927      new_d = true
3928
3929    else
3930      d = nil
3931    end
3932
3933     -- AL << EN/ET/ES      -- W2 + W3 + W6
3934    if last == 'al' and d == 'en' then
3935      d = 'an'            -- W3
3936    elseif last == 'al' and (d == 'et' or d == 'es') then
3937      d = 'on'            -- W6
3938    end
3939
3940    -- EN + CS/ES + EN      -- W4
3941    if d == 'en' and #nodes > 2 then
3942      if (nodes[#nodes -1][2] == 'es' or nodes[#nodes -1][2] == 'cs')
```

147

```
3943         and nodes[#nodes -2][2] == 'en' then
3944       nodes[#nodes -1][2] = 'en'
3945     end
3946   end
3947
3948   -- AN + CS + AN        -- W4 too, because uax9 mixes both cases
3949   if d == 'an' and #nodes > 2 then
3950     if (nodes[#nodes -1][2] == 'cs')
3951         and nodes[#nodes -2][2] == 'an' then
3952       nodes[#nodes -1][2] = 'an'
3953     end
3954   end
3955
3956   -- ET/EN                -- W5 + W7->l / W6->on
3957   if d == 'et' then
3958     first_et = first_et or (#nodes + 1)  -- not yet added to nodes
3959   elseif d == 'en' then
3960     has_en = true
3961     first_et = first_et or (#nodes + 1)  -- not yet added to nodes
3962   elseif first_et then       -- d may be nil here !
3963     if has_en then
3964       if last == 'l' then
3965         temp = 'l'     -- W7
3966       else
3967         temp = 'en'    -- W5
3968       end
3969     else
3970       temp = 'on'      -- W6
3971     end
3972     for e = first_et, #nodes do
3973       if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
3974     end
3975     first_et = nil
3976     has_en = false
3977   end
3978
3979   if d then
3980     if d == 'al' then
3981       d = 'r'
3982       last = 'al'
3983     elseif d == 'l' or d == 'r' then
3984       last = d
3985     end
3986     prev_d = d
3987     table.insert(nodes, {item, d, outer_first})
3988   else
3989     -- Not sure about the following. Looks too 'ad hoc', but it's
3990     -- required for numbers, so that 89 19 becomes 19 89. It also
3991     -- affects n+cs/es+n.
3992     if prev_d == 'an' or prev_d == 'en' then
3993       table.insert(nodes, {item, 'on', nil})
3994     end
3995   end
3996
3997   outer_first = nil
3998
3999 end
4000
4001 -- TODO -- repeated here in case EN/ET is the last node. Find a
```

```
4002   -- better way of doing things:
4003   if first_et then         -- dir may be nil here !
4004     if has_en then
4005       if last == 'l' then
4006         temp = 'l'     -- W7
4007       else
4008         temp = 'en'   -- W5
4009       end
4010     else
4011       temp = 'on'      -- W6
4012     end
4013     for e = first_et, #nodes do
4014       if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
4015     end
4016   end
4017
4018   -- dummy node, to close things
4019   table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
4020
4021   --------------  NEUTRAL -----------------
4022
4023   outer = save_outer
4024   last = outer
4025
4026   local first_on = nil
4027
4028   for q = 1, #nodes do
4029     local item
4030
4031     local outer_first = nodes[q][3]
4032     outer = outer_first or outer
4033     last = outer_first or last
4034
4035     local d = nodes[q][2]
4036     if d == 'an' or d == 'en' then d = 'r' end
4037     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
4038
4039     if d == 'on' then
4040       first_on = first_on or q
4041     elseif first_on then
4042       if last == d then
4043         temp = d
4044       else
4045         temp = outer
4046       end
4047       for r = first_on, q - 1 do
4048         nodes[r][2] = temp
4049         item = nodes[r][1]    -- MIRRORING
4050         if item.id == GLYPH and temp == 'r' then
4051           item.char = characters[item.char].m or item.char
4052         end
4053       end
4054       first_on = nil
4055     end
4056
4057     if d == 'r' or d == 'l' then last = d end
4058   end
4059
4060   --------------  IMPLICIT, REORDER ----------------
```

149

```
4061
4062   outer = save_outer
4063   last = outer
4064
4065   local state = {}
4066   state.has_r = false
4067
4068   for q = 1, #nodes do
4069
4070     local item = nodes[q][1]
4071
4072     outer = nodes[q][3] or outer
4073
4074     local d = nodes[q][2]
4075
4076     if d == 'nsm' then d = last end              -- W1
4077     if d == 'en' then d = 'an' end
4078     local isdir = (d == 'r' or d == 'l')
4079
4080     if outer == 'l' and d == 'an' then
4081       state.san = state.san or item
4082       state.ean = item
4083     elseif state.san then
4084       head, state = insert_numeric(head, state)
4085     end
4086
4087     if outer == 'l' then
4088       if d == 'an' or d == 'r' then      -- im -> implicit
4089         if d == 'r' then state.has_r = true end
4090         state.sim = state.sim or item
4091         state.eim = item
4092       elseif d == 'l' and state.sim and state.has_r then
4093         head, state = insert_implicit(head, state, outer)
4094       elseif d == 'l' then
4095         state.sim, state.eim, state.has_r = nil, nil, false
4096       end
4097     else
4098       if d == 'an' or d == 'l' then
4099         state.sim = state.sim or item
4100         state.eim = item
4101       elseif d == 'r' and state.sim then
4102         head, state = insert_implicit(head, state, outer)
4103       elseif d == 'r' then
4104         state.sim, state.eim = nil, nil
4105       end
4106     end
4107
4108     if isdir then
4109       last = d              -- Don't search back - best save now
4110     elseif d == 'on' and state.san  then
4111       state.san = state.san or item
4112       state.ean = item
4113     end
4114
4115   end
4116
4117   return node.prev(head) or head
4118 end
4119 ⟨/basic⟩
```

## 14 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.

```
4120 ⟨*nil⟩
4121 \ProvidesLanguage{nil}[⟨⟨date⟩⟩ ⟨⟨version⟩⟩ Nil language]
4122 \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.

```
4123 \ifx\l@nohyphenation\@undefined
4124   \@nopatterns{nil}
4125   \adddialect\l@nil0
4126 \else
4127   \let\l@nil\l@nohyphenation
4128 \fi
```

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

```
4129 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}
```

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

`\captionnil`
`\datenil`
```
4130 \let\captionsnil\@empty
4131 \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.

```
4132 \ldf@finish{nil}
4133 ⟨/nil⟩
```

## 15 Support for Plain TeX (`plain.def`)

### 15.1 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 acheive 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`

```
4134 ⟨*bplain | blplain⟩
```

151

```
4135 \catcode`\{=1 % left brace is begin-group character
4136 \catcode`\}=2 % right brace is end-group character
4137 \catcode`\#=6 % hash mark is macro parameter character
```

Now let's see if a file called hyphen.cfg can be found somewhere on TEX's input path by
trying to open it for reading...

```
4138 \openin 0 hyphen.cfg
```

If the file wasn't found the following test turns out true.

```
4139 \ifeof0
4140 \else
```

When hyphen.cfg could be opened we make sure that *it* will be read instead of the file
hyphen.tex which should (according to Don Knuth's ruling) contain the american English
hyphenation patterns and nothing else.
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).

```
4141   \let\a\input
```

Then \input is defined to forget about its argument and load hyphen.cfg instead.

```
4142   \def\input #1 {%
4143     \let\input\a
4144     \a hyphen.cfg
```

Once that's done the original meaning of \input can be restored and the definition of \a
can be forgotten.

```
4145     \let\a\undefined
4146   }
4147 \fi
4148 ⟨/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.

```
4149 ⟨bplain⟩\a plain.tex
4150 ⟨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.

```
4151 ⟨bplain⟩\def\fmtname{babel-plain}
4152 ⟨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.

## 15.2   Emulating some LATEX features

The following code duplicates or emulates parts of LATEX 2ε that are needed for babel.

```
4153 ⟨∗plain⟩
4154 \def\@empty{}
4155 \def\loadlocalcfg#1{%
4156   \openin0#1.cfg
4157   \ifeof0
4158     \closein0
4159   \else
4160     \closein0
4161     {\immediate\write16{**********************************}%
4162     \immediate\write16{* Local config file #1.cfg used}%
4163     \immediate\write16{*}%
```

```
4164        }
4165    \input #1.cfg\relax
4166   \fi
4167   \@endofldf}
```

## 15.3  General tools

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

```
4168 \long\def\@firstofone#1{#1}
4169 \long\def\@firstoftwo#1#2{#1}
4170 \long\def\@secondoftwo#1#2{#2}
4171 \def\@nnil{\@nil}
4172 \def\@gobbletwo#1#2{}
4173 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
4174 \def\@star@or@long#1{%
4175   \@ifstar
4176   {\let\l@ngrel@x\relax#1}%
4177   {\let\l@ngrel@x\long#1}}
4178 \let\l@ngrel@x\relax
4179 \def\@car#1#2\@nil{#1}
4180 \def\@cdr#1#2\@nil{#2}
4181 \let\@typeset@protect\relax
4182 \let\protected@edef\edef
4183 \long\def\@gobble#1{}
4184 \edef\@backslashchar{\expandafter\@gobble\string\\}
4185 \def\strip@prefix#1>{}
4186 \def\g@addto@macro#1#2{{%
4187   \toks@\expandafter{#1#2}%
4188   \xdef#1{\the\toks@}}}
4189 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
4190 \def\@nameuse#1{\csname #1\endcsname}
4191 \def\@ifundefined#1{%
4192   \expandafter\ifx\csname#1\endcsname\relax
4193     \expandafter\@firstoftwo
4194   \else
4195     \expandafter\@secondoftwo
4196   \fi}
4197 \def\@expandtwoargs#1#2#3{%
4198   \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
4199 \def\zap@space#1 #2{%
4200   #1%
4201   \ifx#2\@empty\else\expandafter\zap@space\fi
4202   #2}
```

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

```
4203 \ifx\@preamblecmds\@undefined
4204   \def\@preamblecmds{}
4205 \fi
4206 \def\@onlypreamble#1{%
4207   \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
4208     \@preamblecmds\do#1}}
4209 \@onlypreamble\@onlypreamble
```

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

```
4210 \def\begindocument{%
4211   \@begindocumenthook
```

```
4212 \global\let\@begindocumenthook\@undefined
4213 \def\do##1{\global\let##1\@undefined}%
4214 \@preamblecmds
4215 \global\let\do\noexpand}

4216 \ifx\@begindocumenthook\@undefined
4217   \def\@begindocumenthook{}
4218 \fi
4219 \@onlypreamble\@begindocumenthook
4220 \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.

```
4221 \def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
4222 \@onlypreamble\AtEndOfPackage
4223 \def\@endofldf{}
4224 \@onlypreamble\@endofldf
4225 \let\bbl@afterlang\@empty
4226 \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.

```
4227 \ifx\if@filesw\@undefined
4228   \expandafter\let\csname if@filesw\expandafter\endcsname
4229     \csname iffalse\endcsname
4230 \fi
```

Mimick LaTeX's commands to define control sequences.

```
4231 \def\newcommand{\@star@or@long\new@command}
4232 \def\new@command#1{%
4233   \@testopt{\@newcommand#1}0}
4234 \def\@newcommand#1[#2]{%
4235   \@ifnextchar [{\@xargdef#1[#2]}%
4236                {\@argdef#1[#2]}}
4237 \long\def\@argdef#1[#2]#3{%
4238   \@yargdef#1\@ne{#2}{#3}}
4239 \long\def\@xargdef#1[#2][#3]#4{%
4240   \expandafter\def\expandafter#1\expandafter{%
4241     \expandafter\@protected@testopt\expandafter #1%
4242     \csname\string#1\expandafter\endcsname{#3}}%
4243   \expandafter\@yargdef \csname\string#1\endcsname
4244   \tw@{#2}{#4}}
4245 \long\def\@yargdef#1#2#3{%
4246   \@tempcnta#3\relax
4247   \advance \@tempcnta \@ne
4248   \let\@hash@\relax
4249   \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
4250   \@tempcntb #2%
4251   \@whilenum\@tempcntb <\@tempcnta
4252   \do{%
4253     \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
4254     \advance\@tempcntb \@ne}%
4255   \let\@hash@##%
4256   \l@ngrel@x\expandafter\def\expandafter#1\reserved@a}
4257 \def\providecommand{\@star@or@long\provide@command}
4258 \def\provide@command#1{%
4259   \begingroup
4260     \escapechar\m@ne\xdef\@gtempa{{\string#1}}%
4261   \endgroup
```

154

```
4262    \expandafter\@ifundefined\@gtempa
4263      {\def\reserved@a{\new@command#1}}%
4264      {\let\reserved@a\relax
4265       \def\reserved@a{\new@command\reserved@a}}%
4266    \reserved@a}%
4267 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
4268 \def\declare@robustcommand#1{%
4269    \edef\reserved@a{\string#1}%
4270    \def\reserved@b{#1}%
4271    \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
4272    \edef#1{%
4273      \ifx\reserved@a\reserved@b
4274        \noexpand\x@protect
4275        \noexpand#1%
4276      \fi
4277      \noexpand\protect
4278      \expandafter\noexpand\csname\bbl@stripslash#1 \endcsname
4279    }%
4280    \expandafter\new@command\csname\bbl@stripslash#1 \endcsname
4281 }
4282 \def\x@protect#1{%
4283    \ifx\protect\@typeset@protect\else
4284      \@x@protect#1%
4285    \fi
4286 }
4287 \def\@x@protect#1\fi#2#3{%
4288    \fi\protect#1%
4289 }
```

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.

```
4290 \def\bbl@tempa{\csname newif\endcsname\ifin@}
4291 \ifx\in@\@undefined
4292   \def\in@#1#2{%
4293     \def\in@@##1#1##2##3\in@@{%
4294       \ifx\in@##2\in@false\else\in@true\fi}%
4295     \in@@#2#1\in@\in@@}
4296 \else
4297   \let\bbl@tempa\@empty
4298 \fi
4299 \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).

```
4300 \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.

```
4301 \def\@ifl@aded#1#2#3#4{}
```

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

```
4302 \ifx\@tempcnta\@undefined
4303   \csname newcount\endcsname\@tempcnta\relax
4304 \fi
4305 \ifx\@tempcntb\@undefined
4306   \csname newcount\endcsname\@tempcntb\relax
4307 \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).

```
4308 \ifx\bye\@undefined
4309   \advance\count10 by -2\relax
4310 \fi
4311 \ifx\@ifnextchar\@undefined
4312   \def\@ifnextchar#1#2#3{%
4313     \let\reserved@d=#1%
4314     \def\reserved@a{#2}\def\reserved@b{#3}%
4315     \futurelet\@let@token\@ifnch}
4316   \def\@ifnch{%
4317     \ifx\@let@token\@sptoken
4318       \let\reserved@c\@xifnch
4319     \else
4320       \ifx\@let@token\reserved@d
4321         \let\reserved@c\reserved@a
4322       \else
4323         \let\reserved@c\reserved@b
4324       \fi
4325     \fi
4326     \reserved@c}
4327   \def\:{\let\@sptoken= } \:  % this makes \@sptoken a space token
4328   \def\:{\@xifnch} \expandafter\def\: {\futurelet\@let@token\@ifnch}
4329 \fi
4330 \def\@testopt#1#2{%
4331   \@ifnextchar[{#1}{#1[#2]}}
4332 \def\@protected@testopt#1{%
4333   \ifx\protect\@typeset@protect
4334     \expandafter\@testopt
4335   \else
4336     \@x@protect#1%
4337   \fi}
4338 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
4339     #2\relax}\fi}
4340 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
4341       \else\expandafter\@gobble\fi{#1}}
```

## 15.4   Encoding related macros

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

```
4342 \def\DeclareTextCommand{%
4343   \@dec@text@cmd\providecommand
4344 }
4345 \def\ProvideTextCommand{%
4346   \@dec@text@cmd\providecommand
4347 }
4348 \def\DeclareTextSymbol#1#2#3{%
4349   \@dec@text@cmd\chardef#1{#2}#3\relax
4350 }
4351 \def\@dec@text@cmd#1#2#3{%
4352   \expandafter\def\expandafter#2%
```

```
4353        \expandafter{%
4354           \csname#3-cmd\expandafter\endcsname
4355           \expandafter#2%
4356           \csname#3\string#2\endcsname
4357        }%
4358 %    \let\@ifdefinable\@rc@ifdefinable
4359     \expandafter#1\csname#3\string#2\endcsname
4360 }
4361 \def\@current@cmd#1{%
4362   \ifx\protect\@typeset@protect\else
4363       \noexpand#1\expandafter\@gobble
4364   \fi
4365 }
4366 \def\@changed@cmd#1#2{%
4367   \ifx\protect\@typeset@protect
4368       \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
4369         \expandafter\ifx\csname ?\string#1\endcsname\relax
4370           \expandafter\def\csname ?\string#1\endcsname{%
4371             \@changed@x@err{#1}%
4372           }%
4373         \fi
4374         \global\expandafter\let
4375           \csname\cf@encoding \string#1\expandafter\endcsname
4376           \csname ?\string#1\endcsname
4377       \fi
4378     \csname\cf@encoding\string#1%
4379         \expandafter\endcsname
4380   \else
4381       \noexpand#1%
4382   \fi
4383 }
4384 \def\@changed@x@err#1{%
4385    \errhelp{Your command will be ignored, type <return> to proceed}%
4386    \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
4387 \def\DeclareTextCommandDefault#1{%
4388   \DeclareTextCommand#1?%
4389 }
4390 \def\ProvideTextCommandDefault#1{%
4391   \ProvideTextCommand#1?%
4392 }
4393 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
4394 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
4395 \def\DeclareTextAccent#1#2#3{%
4396   \DeclareTextCommand#1{#2}[1]{\accent#3 ##1}
4397 }
4398 \def\DeclareTextCompositeCommand#1#2#3#4{%
4399   \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
4400   \edef\reserved@b{\string##1}%
4401   \edef\reserved@c{%
4402     \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
4403   \ifx\reserved@b\reserved@c
4404     \expandafter\expandafter\expandafter\ifx
4405       \expandafter\@car\reserved@a\relax\relax\@nil
4406       \@text@composite
4407     \else
4408       \edef\reserved@b##1{%
4409         \def\expandafter\noexpand
4410           \csname#2\string#1\endcsname####1{%
4411           \noexpand\@text@composite
```

157

```
4412                \expandafter\noexpand\csname#2\string#1\endcsname
4413                ####1\noexpand\@empty\noexpand\@text@composite
4414                {##1}%
4415            }%
4416          }%
4417          \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
4418      \fi
4419      \expandafter\def\csname\expandafter\string\csname
4420          #2\endcsname\string#1-\string#3\endcsname{#4}
4421    \else
4422      \errhelp{Your command will be ignored, type <return> to proceed}%
4423      \errmessage{\string\DeclareTextCompositeCommand\space used on
4424          inappropriate command \protect#1}
4425    \fi
4426 }
4427 \def\@text@composite#1#2#3\@text@composite{%
4428    \expandafter\@text@composite@x
4429        \csname\string#1-\string#2\endcsname
4430 }
4431 \def\@text@composite@x#1#2{%
4432    \ifx#1\relax
4433        #2%
4434    \else
4435        #1%
4436    \fi
4437 }
4438 %
4439 \def\@strip@args#1:#2-#3\@strip@args{#2}
4440 \def\DeclareTextComposite#1#2#3#4{%
4441    \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
4442    \bgroup
4443        \lccode`\@=#4%
4444        \lowercase{%
4445    \egroup
4446        \reserved@a @%
4447    }%
4448 }
4449 %
4450 \def\UseTextSymbol#1#2{%
4451 %    \let\@curr@enc\cf@encoding
4452 %    \@use@text@encoding{#1}%
4453    #2%
4454 %    \@use@text@encoding\@curr@enc
4455 }
4456 \def\UseTextAccent#1#2#3{%
4457 %    \let\@curr@enc\cf@encoding
4458 %    \@use@text@encoding{#1}%
4459 %    #2{\@use@text@encoding\@curr@enc\selectfont#3}%
4460 %    \@use@text@encoding\@curr@enc
4461 }
4462 \def\@use@text@encoding#1{%
4463 %    \edef\f@encoding{#1}%
4464 %    \xdef\font@name{%
4465 %        \csname\curr@fontshape/\f@size\endcsname
4466 %    }%
4467 %    \pickup@font
4468 %    \font@name
4469 %    \@@enc@update
4470 }
```

```
4471 \def\DeclareTextSymbolDefault#1#2{%
4472     \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
4473 }
4474 \def\DeclareTextAccentDefault#1#2{%
4475     \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
4476 }
4477 \def\cf@encoding{OT1}
```

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

```
4478 \DeclareTextAccent{\"}{OT1}{127}
4479 \DeclareTextAccent{\'}{OT1}{19}
4480 \DeclareTextAccent{\^}{OT1}{94}
4481 \DeclareTextAccent{\`}{OT1}{18}
4482 \DeclareTextAccent{\~}{OT1}{126}
```

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

```
4483 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
4484 \DeclareTextSymbol{\textquotedblright}{OT1}{`\"}
4485 \DeclareTextSymbol{\textquoteleft}{OT1}{`\`}
4486 \DeclareTextSymbol{\textquoteright}{OT1}{`\'}
4487 \DeclareTextSymbol{\i}{OT1}{16}
4488 \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`.

```
4489 \ifx\scriptsize\@undefined
4490     \let\scriptsize\sevenrm
4491 \fi
4492 ⟨/plain⟩
```

## 16   Acknowledgements

I would like to thank all who volunteered as $\beta$-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]  Donald E. Knuth, *The TeXbook,* Addison-Wesley, 1986.

[3]  Leslie Lamport, *LaTeX, A document preparation System*, Addison-Wesley, 1986.

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

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

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

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

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

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

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