Babel

Original author
Johannes L. Braams

Current maintainer
Javier Bezos

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

3

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 on `https://github.com/latex3/babel/issues`, which is better than just complaining on an e-mail list or a web forum.

- 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/babel` (which provides some sample files, too).

- See section 3.1 for contributing a language.

- The first sections describe the traditional way of loading a language (with `ldf` files). The alternative way based on `ini` files, which complements the previous one (it will *not* replace it), is described below.

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

```
\documentclass{article}

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

\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}
```

NOTE Some classes load babel with a hardcoded language option. Sometimes, the main language could be overridden with something like that before `\documentclass`:

```
    \PassOptionsToPackage{main=english}{babel}
```

WARNING Languages may be set as global and as package option at the same time, but in such a case you should set explicitly the main language with the package option `main`:

```
    \documentclass[italian]{book}
    \usepackage[ngerman,main=italian]{babel}
```

WARNING In the preamble the main language has *not* been selected, except hyphenation patterns and the name assigned to \languagename (in particular, shorthands, captions and date are not activated). If you need to define boxes and the like in the preamble, you might want to use some of the language selectors described below.

To switch the language there are two basic macros, described below in detail: \selectlanguage is used for blocks of text, while \foreignlanguage is for chunks of text inside paragraphs.

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

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

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.

```
\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}.
```

---

[2]In old versions the error read "You have used an old interface to call babel", not very helpful.

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

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

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

1.6 Plain

In Plain, load languages styles with `\input` and then use `\begindocument` (the latter is defined by babel):

```
\input estonian.sty
\begindocument
```

WARNING Not all languages provide a `sty` file and some of them are not compatible with Plain.[4]

1.7 Basic language selectors

This section describes the commands to be used in the document to switch the language in multilingual documents. In most cases, only the two basic macros `\selectlanguage` and `\foreignlanguage` are necessary. The environments `otherlanguage`, `otherlanguage*` and `hyphenrules` are auxiliary, and described in the next section.
The main language is selected automatically when the document environment begins.

`\selectlanguage`   {⟨*language*⟩}

When a user wants to switch from one language to another he can do so using the macro `\selectlanguage`. This macro takes the language, defined previously by a language definition file, as its argument. It calls several macros that should be defined in the language definition files to activate the special definitions for the language chosen:

```
\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:

---

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

```
{\selectlanguage{<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). With the `bidi` option, it also enters in horizontal mode (this is not done always for backwards compatibility).

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 typesetting with right-to-left typesetting in engines not supporting a change in the writing direction inside a line. However, by default it never complied with the documented behavior and it is just a version as environment of `\foreignlanguage`, except when the option `bidi` is set – in this case, `\foreignlanguage` emits a `\leavevmode`, while `otherlanguage*` does not.

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

The environment `hyphenrules` can be used to select *only* the hyphenation rules to be used (it can be used as command, too). This can for instance be used to select 'nohyphenation', provided that in `language.dat` the 'language' nohyphenation is defined by loading `zerohyph.tex`. It deactivates language shorthands, too (but not user shorthands).
Except for these simple uses, `hyphenrules` is discouraged and `otherlanguage*` (the starred version) is preferred, as the former does not take into account possible changes in

encodings of characters like, say, ' done by some languages (eg, italian, french, ukraineb). To set hyphenation exceptions, use \babelhyphenation (see below).

1.9   More on selection

\babeltags   {⟨*tag1*⟩ = ⟨*language1*⟩, ⟨*tag2*⟩ = ⟨*language2*⟩, ...}

New 3.9i In multilingual documents with many language switches the commands above can be cumbersome. With this tool shorter names can be defined. It adds nothing really new – it is just syntactical sugar.
It defines \text⟨*tag1*⟩{⟨*text*⟩} to be \foreignlanguage{⟨*language1*⟩}{⟨*text*⟩}, and \begin{⟨*tag1*⟩} to be \begin{otherlanguage*}{⟨*language1*⟩}, and so on. Note \⟨*tag1*⟩ is also allowed, but remember to set it locally inside a group.

EXAMPLE With

```
\babeltags{de = german}
```

you can write

```
text \textde{German text} text
```

and

```
text
\begin{de}
  German text
\end{de}
text
```

NOTE Something like \babeltags{finnish = finnish} is legitimate – it defines \textfinnish and \finnish (and, of course, \begin{finnish}).

NOTE Actually, there may be another advantage in the 'short' syntax \text⟨*tag*⟩, namely, it is not affected by \MakeUppercase (while \foreignlanguage is).

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

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

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

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

```
\babelensure{polish}
```

By default only the basic captions and `\today` are redefined, but you can add further macros with the key `include` in the optional argument (without commas). Macros not to be modified are listed in `exclude`. You can also enforce a font encoding with `fontenc`.[5] A couple of examples:

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

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

1.10 Shorthands

A *shorthand* is a sequence of one or two characters that expands to arbitrary TeX code. Shorthands can be used for different kinds of things, 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*⟩}

11

<div style="text-align:right">\shorthandoff</div> `*`{⟨*shorthands-list*⟩}

It is sometimes necessary to switch a shorthand character off temporarily, because it must be used in an entirely different way. For this purpose, the user commands `\shorthandoff` and `\shorthandon` are provided. They each take a list of characters as their arguments. The command `\shorthandoff` sets the `\catcode` for each of the characters in its argument to other (12); the command `\shorthandon` sets the `\catcode` to active (13). Both commands only work on 'known' shorthand characters.

New 3.9a   However, `\shorthandoff` does not behave as you would expect with characters like ~ or ^, because they usually are not "other". For them `\shorthandoff*` is provided, so that with

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

~ is still active, very likely with the meaning of a non-breaking space, and ^ is the superscript character. The catcodes used are those when the shorthands are defined, usually when language files are loaded.

<div style="text-align:right">\useshorthands</div> `*`{⟨*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.

<div style="text-align:right">\defineshorthand</div> [⟨*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:

---

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

```
\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

Languages with only " as defined shorthand character   Albanian, Bulgarian, Danish, Dutch, Finnish, German (old and new orthography, also Austrian), Icelandic, Italian, Norwegian, Polish, Portuguese (also Brazilian), Russian, Serbian (with Latin script), Slovene, Swedish, Ukrainian, Upper Sorbian

Basque   "   '   ~
Breton   :   ;   ?   !
Catalan   "   '   `
Czech   "   -
Esperanto   ^
Estonian   "   ~
French (all varieties) :   ;   ?   !
Galician   "   .   '   ~   <   >
Greek   ~
Hungarian   `
Kurmanji   ^
Latin   "   ^   =
Slovak   "   ^   '   -
Spanish   "   .   <   >   '
Turkish   :   !   =

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

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

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

1.11   Package options

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

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

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

---

[6]Actually, any name not corresponding to a language group does the same as none. However, follow this convention because it might be enforced in future releases of babel to catch possible errors.
[7]Thanks to Enrico Gregorio
[8]This declaration serves to nothing, but it is preserved for backward compatibility.

activegrave	Same for `.

shorthands=	⟨*char*⟩⟨*char*⟩... | off

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

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

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

safe=	none | ref | bib

Some LaTeX macros are redefined so that using shorthands is safe. With `safe=bib` only `\nocite`, `\bibcite` and `\bibitem` are redefined. With `safe=ref` only `\newlabel`, `\ref` and `\pageref` are redefined (as well as a few macros from varioref and ifthen). With `safe=none` no macro is redefined. This option is strongly recommended, because a good deal of incompatibilities and errors are related to these redefinitions – 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*`

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=	`default` | `basic` | `basic-r`

New 3.14 Selects the bidi algorithm to be used in luatex and xetex. See sec. 1.21.

layout=	

New 3.16 Selects which layout elements are adapted in bidi documents. See sec. 1.21.

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*⟩}

---

[9] You can use alternatively the package silence.
[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.

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

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

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

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

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

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 (in other words, \babelprovide is mainly intended for auxiliary tasks).

EXAMPLE Although Georgian has its own ldf file, here is how to declare this language with an ini file in Unicode engines.

```
\documentclass{book}

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

\babelfont{rm}{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):

Code	Language	Code	Language
af	Afrikaans[ul]	es	Spanish[ul]
agq	Aghem	et	Estonian[ul]
ak	Akan	eu	Basque[ul]
am	Amharic[ul]	ewo	Ewondo
ar	Arabic[ul]	fa	Persian[ul]
ar-DZ	Arabic[ul]	ff	Fulah
ar-MA	Arabic[ul]	fi	Finnish[ul]
ar-SY	Arabic[ul]	fil	Filipino
as	Assamese	fo	Faroese
asa	Asu	fr	French[ul]
ast	Asturian[ul]	fr-BE	French[ul]
az-Cyrl	Azerbaijani	fr-CA	French[ul]
az-Latn	Azerbaijani	fr-CH	French[ul]
az	Azerbaijani[ul]	fr-LU	French[ul]
bas	Basaa	fur	Friulian[ul]
be	Belarusian[ul]	fy	Western Frisian
bem	Bemba	ga	Irish[ul]
bez	Bena	gd	Scottish Gaelic[ul]
bg	Bulgarian[ul]	gl	Galician[ul]
bm	Bambara	gsw	Swiss German
bn	Bangla[ul]	gu	Gujarati
bo	Tibetan[u]	guz	Gusii
brx	Bodo	gv	Manx
bs-Cyrl	Bosnian	ha-GH	Hausa
bs-Latn	Bosnian[ul]	ha-NE	Hausa[l]
bs	Bosnian[ul]	ha	Hausa
ca	Catalan[ul]	haw	Hawaiian
ce	Chechen	he	Hebrew[ul]
cgg	Chiga	hi	Hindi[u]
chr	Cherokee	hr	Croatian[ul]
ckb	Central Kurdish	hsb	Upper Sorbian[ul]
cs	Czech[ul]	hu	Hungarian[ul]
cy	Welsh[ul]	hy	Armenian
da	Danish[ul]	ia	Interlingua[ul]
dav	Taita	id	Indonesian[ul]
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

kn	Kannada[ul]	pl	Polish[ul]
ko	Korean	pms	Piedmontese[ul]
kok	Konkani	ps	Pashto
ks	Kashmiri	pt-BR	Portuguese[ul]
ksb	Shambala	pt-PT	Portuguese[ul]
ksf	Bafia	pt	Portuguese[ul]
ksh	Colognian	qu	Quechua
kw	Cornish	rm	Romansh[ul]
ky	Kyrgyz	rn	Rundi
lag	Langi	ro	Romanian[ul]
lb	Luxembourgish	rof	Rombo
lg	Ganda	ru	Russian[ul]
lkt	Lakota	rw	Kinyarwanda
ln	Lingala	rwk	Rwa
lo	Lao[ul]	sa-Beng	Sanskrit
lrc	Northern Luri	sa-Deva	Sanskrit
lt	Lithuanian[ul]	sa-Gujr	Sanskrit
lu	Luba-Katanga	sa-Knda	Sanskrit
luo	Luo	sa-Mlym	Sanskrit
luy	Luyia	sa-Telu	Sanskrit
lv	Latvian[ul]	sa	Sanskrit
mas	Masai	sah	Sakha
mer	Meru	saq	Samburu
mfe	Morisyen	sbp	Sangu
mg	Malagasy	se	Northern Sami[ul]
mgh	Makhuwa-Meetto	seh	Sena
mgo	Meta'	ses	Koyraboro Senni
mk	Macedonian[ul]	sg	Sango
ml	Malayalam[ul]	shi-Latn	Tachelhit
mn	Mongolian	shi-Tfng	Tachelhit
mr	Marathi[ul]	shi	Tachelhit
ms-BN	Malay[l]	si	Sinhala
ms-SG	Malay[l]	sk	Slovak[ul]
ms	Malay[ul]	sl	Slovenian[ul]
mt	Maltese	smn	Inari Sami
mua	Mundang	sn	Shona
my	Burmese	so	Somali
mzn	Mazanderani	sq	Albanian[ul]
naq	Nama	sr-Cyrl-BA	Serbian[ul]
nb	Norwegian Bokmål[ul]	sr-Cyrl-ME	Serbian[ul]
nd	North Ndebele	sr-Cyrl-XK	Serbian[ul]
ne	Nepali	sr-Cyrl	Serbian[ul]
nl	Dutch[ul]	sr-Latn-BA	Serbian[ul]
nmg	Kwasio	sr-Latn-ME	Serbian[ul]
nn	Norwegian Nynorsk[ul]	sr-Latn-XK	Serbian[ul]
nnh	Ngiemboon	sr-Latn	Serbian[ul]
nus	Nuer	sr	Serbian[ul]
nyn	Nyankole	sv	Swedish[ul]
om	Oromo	sw	Swahili
or	Odia	ta	Tamil[u]
os	Ossetic	te	Telugu[ul]
pa-Arab	Punjabi	teo	Teso
pa-Guru	Punjabi	th	Thai[ul]
pa	Punjabi	ti	Tigrinya

tk	Turkmen[ul]	wae	Walser
to	Tongan	xog	Soga
tr	Turkish[ul]	yav	Yangben
twq	Tasawaq	yi	Yiddish
tzm	Central Atlas Tamazight	yo	Yoruba
ug	Uyghur	yue	Cantonese
uk	Ukrainian[ul]	zgh	Standard Moroccan
ur	Urdu[ul]		Tamazight
uz-Arab	Uzbek	zh-Hans-HK	Chinese
uz-Cyrl	Uzbek	zh-Hans-MO	Chinese
uz-Latn	Uzbek	zh-Hans-SG	Chinese
uz	Uzbek	zh-Hans	Chinese
vai-Latn	Vai	zh-Hant-HK	Chinese
vai-Vaii	Vai	zh-Hant-MO	Chinese
vai	Vai	zh-Hant	Chinese
vi	Vietnamese[ul]	zh	Chinese
vun	Vunjo	zu	Zulu

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

aghem	bemba
akan	bena
albanian	bengali
american	bodo
amharic	bosnian-cyrillic
arabic	bosnian-cyrl
arabic-algeria	bosnian-latin
arabic-DZ	bosnian-latn
arabic-morocco	bosnian
arabic-MA	brazilian
arabic-syria	breton
arabic-SY	british
armenian	bulgarian
assamese	burmese
asturian	canadian
asu	cantonese
australian	catalan
austrian	centralatlastamazight
azerbaijani-cyrillic	centralkurdish
azerbaijani-cyrl	chechen
azerbaijani-latin	cherokee
azerbaijani-latn	chiga
azerbaijani	chinese-hans-hk
bafia	chinese-hans-mo
bambara	chinese-hans-sg
basaa	chinese-hans
basque	chinese-hant-hk
belarusian	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
sanskrit-beng
sanskrit-bengali
sanskrit-deva
sanskrit-devanagari
sanskrit-gujarati
sanskrit-gujr
sanskrit-kannada
sanskrit-knda
sanskrit-malayalam
sanskrit-mlym
sanskrit-telu
sanskrit-telugu
sanskrit
scottishgaelic
sena
serbian-cyrillic-bosniaherzegovina
serbian-cyrillic-kosovo
serbian-cyrillic-montenegro
serbian-cyrillic
serbian-cyrl-ba
serbian-cyrl-me
serbian-cyrl-xk
serbian-cyrl
serbian-latin-bosniaherzegovina
serbian-latin-kosovo
serbian-latin-montenegro
serbian-latin
serbian-latn-ba
serbian-latn-me
serbian-latn-xk
serbian-latn
serbian
shambala
shona
sichuanyi
sinhala
slovak
slovene
slovenian
soga
somali
spanish-mexico
spanish-mx

spanish
standardmoroccantamazight
swahili
swedish
swissgerman
tachelhit-latin
tachelhit-latn
tachelhit-tfng
tachelhit-tifinagh
tachelhit
taita
tamil
tasawaq
telugu
teso
thai
tibetan
tigrinya
tongan
turkish
turkmen
ukenglish
ukrainian
uppersorbian
urdu
usenglish

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

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}
```

---

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

```
    \babelprovide[import]{hebrew}

    \babelfont{rm}{FreeSerif}

    \begin{document}

    Svenska \foreignlanguage{hebrew}{עִבְרִית} svenska.

    \end{document}
```

If on the other hand you have to resort to different fonts, you 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 families.

EXAMPLE Here is how to do it:

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

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

NOTE You may load fontspec explicitly. For example:

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

This makes sure the OpenType script for Devanagari is deva and not dev2 (luatex does not detect automatically the correct script[14]). You may also pass some options to fontspec: with silent, the warnings about unavailable scripts or languages are not shown (they are only really useful when the document format is being set up).

NOTE Directionality is a property affecting margins, indentation, column order, etc., not just text. Therefore, it is under the direct control of the language, which applies both the script and the direction to the text. As a consequence, there is no need to set Script when declaring a font (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).

---

[14]And even with the correct code some fonts could be rendered incorrectly by fontspec, so double check the results. xetex fares better, but some font are still problematic.

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.

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

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

first loads danish.ldf, and then redefines the captions for danish (as provided by the ini file) and prevents hyphenation. The rest of the language definitions are not touched.

1.16   Creating a language

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

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

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*⟩

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.
It may be used without a value. In such a case, the `ini` file set in the corresponding `babel-<language>.tex` (where `<language>` is the last argument in `\babelprovide`) is imported. See the list of recognized languages above. So, the previous example could be written:

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

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` 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, this option defines an additional command for dates: \<language>date, which takes three arguments, namely, year, month and day numbers. In fact, \today calls \<language>today, which in turn calls \<language>date{\the\year}{\the\month}{\the\day}.

captions= ⟨*language-tag*⟩

Loads only the strings. For example:

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

hyphenrules= ⟨*language-list*⟩

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

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

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

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

In other engines it just suppresses hyphenation (because the pattern list is empty).

main This valueless option makes the language the main one. 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, so you must use it if for some reason the default value is wrong.

language= ⟨*language-name*⟩

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

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

mapfont= direction

Assigns the font for the writing direction of this language.[15] More precisely, what mapfont=direction means is, 'when a character has the same direction as the script for

---

[15]There will be another value, language, not yet implemented.

the "provided" language, then change its font to that set for this language'. There are 3 directions, following the bidi Unicode algorithm, namely, Arabic-like, Hebrew-like and left to right.[16] So, there should be at most 3 directives of this kind.

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

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

intrapenalty= ⟨*penalty*⟩

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

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 the default in `ini`-based languages).

1.17 Digits

New 3.20 About thirty `ini` files define a field named `digits.native`. When it is present, two macros are created: `\<language>digits` and `\<language>counter` (only xetex and luatex). With the first, a string of 'Latin' digits are converted to the native digits of that language; the second takes a counter name as argument. With the option `maparabic` in `\babelprovide`, `\arabic` is redefined to produce the native digits (this is done *globally*, to avoid inconsistencies in, for example, page numbering).
For example:

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

Languages providing native digits in all or some variants are *ar, as, bn, bo, brx, ckb, dz, fa, gu, hi, km, kn, kok, ks, lo, lrc, ml, mr, my, mzn, ne, or, pa, ps, ta, te, th, ug, ur, uz, vai, yue, zh.*

1.18 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

---

[16]In future releases an new value (`script`) will be added.

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.

1.19   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 opportunity or, in TeX terms, a "discretionary"; a *hard hyphen* is a hyphen with a breaking opportunity after it. A further type is a *non-breaking hyphen*, a hyphen without a breaking opportunity.
In TeX, - and \- forbid further breaking opportunities in the word. This is the desired behavior very often, but not always, and therefore many languages provide shorthands for these cases. Unfortunately, this has not been done consistently: for example, "- in Dutch, 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 opportunities 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.

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

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

New 3.9m *In luatex only,*[17] 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.

1.20 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.[18]
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.[19]

\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

---

[17]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.
[18]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.
[19]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 (they are stored in `\BabelNonText`, used in some special cases when no Latin encoding is explicitly set).

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

1.21   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**   The current code for **text** in luatex should be considered essentially stable, but, of course, it is not bug free and there could be improvements in the future, because setting bidi text has many subtleties (see for example <https://www.w3.org/TR/html-bidi/>). A basic stable version for other engines must wait very likely until (Northern) Winter. This applies to text, but **graphical** elements, including the `picture` environment and PDF or PS based graphics, are not yet correctly handled (far from trivial). Also, indexes and the like are under study, as well as math.
>
> 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 | 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 in typical cases.
 New 3.19   Finally, `basic` supports both L and R text. (They are named `basic` mainly because they only consider the intrinsic direction of scripts and weak directionality.) There are samples on GitHub, under `/required/babel/samples`. See particularly `lua-bidibasic.tex` and `lua-secenum.tex`.

> **EXAMPLE**   The following text comes from the Arabic Wikipedia (article about Arabia). Copy-pasting some text from the Wikipedia is a good way to test this feature. Remember `basic-r` is available in luatex only.[20]

---

[20]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. Have a look at the workaround available on GitHub, under `/required/babel/samples`

```
\documentclass{article}

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

\babelprovide[import, main]{arabic}

\babelfont{rm}{FreeSerif}

\begin{document}

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

\end{document}
```

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

```
\documentclass{book}

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

\babelprovide[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-'aṣr} (MSA) and
فصحى التراث \textit{fuṣḥā t-turāth} (CA).

\end{document}
```

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

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

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

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

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

New 3.16 *To be expanded.* Selects which layout elements are adapted in bidi documents, including some text elements. You may use several options with a dot-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 (except luatex with `bidi=basic`) to reorder section numbers and the like (eg, ⟨*subsection*⟩.⟨*section*⟩); required in xetex and pdftex for counters in general, as well as in luatex with `bidi=default`; required in luatex for numeric footnote marks >9 with `bidi=basic-r` (but *not* with `bidi=basic`); note, however, it could depend on the counter format.

With counters, \arabic is not only considered L text always (with \babelsublr, see below), but also an "isolated" block which does not interact with the surrounding chars. So, while 1.2 in R text is rendered in that order with `bidi=basic` (as a decimal number), in \arabic{c1}.\arabic{c2} the visual order is *c2.c1*. Of course, you may always adjust the order by changing the language, if necessary.[21]

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 ,

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

\babelsublr   {⟨*lr-text*⟩}

Digits in pdftex must be marked up explicitly (unlike luatex with `bidi=basic` or `bidi=basic-r` and, usually, xetex). This command is provided to set {⟨*lr-text*⟩} in L mode if necessary. It's intended for what Unicode calls weak characters, because words are best set with the corresponding language. For this reason, there is no rl counterpart. Any \babelsublr in *explicit* L mode is ignored. However, with `bidi=basic` and *implicit* L, it first returns to R and then switches to explicit L. To clarify this point, consider, in an R context:

---

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

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

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

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

\BabelPatchSection   {⟨*section-name*⟩}

Mainly for bidi text, but it 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 (it also "isolates" the page number in heads, for a proper bidi behavior), but with this command you can set them individually if necessary (but note then tocs and marks are not touched).

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

New 3.17   Something like:

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

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

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

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

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

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

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

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

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

1.22   Language attributes

\languageattribute This is a user-level command, to be used in the preamble of a document (after
\usepackage[...]{babel}), that declares which attributes are to be used for a given
language. It takes two arguments: the first is the name of the language; the second, a (list
of) attribute(s) to be used. Attributes must be set in the preamble and only once – they
cannot be turned on and off. The command checks whether the language is known in this
document and whether the attribute(s) are known for this language.

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

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

1.23   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).

1.24 Languages supported by babel with ldf files

In the following table most of the languages supported by babel with and `.ldf` file are listed, together with the names of the option which you can load babel with for each language. Note this list is open and the current options may be different. It does not include `ini` files.

Afrikaans afrikaans
Azerbaijani azerbaijani
Basque basque
Breton breton
Bulgarian bulgarian
Catalan catalan
Croatian croatian
Czech czech
Danish danish
Dutch dutch
English english, USenglish, american, UKenglish, british, canadian, australian, newzealand
Esperanto esperanto
Estonian estonian
Finnish finnish
French french, francais, canadien, acadian
Galician galician
German austrian, german, germanb, ngerman, naustrian
Greek greek, polutonikogreek
Hebrew hebrew
Icelandic icelandic
Indonesian 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.

NOTE Please, for info about the support in luatex for some complex scripts, see the wiki, on `https://github.com/latex3/latex2e/wiki/Babel:-Remarks-on-the-luatex-support-for-some-scripts`.

1.25   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.[22] 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).

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

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

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

1.26 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).

---

[22]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.

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. Also on the roadmap are better support for R layouts (lists, footnotes, tables, column order), page and section numbering, and maybe kashida justification.

Useful additions would be, for example, time, currency, addresses and personal names.[23]. 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.

1.27 Tentative and experimental code

See the code section for `\foreignlanguage*` (a new starred version of `\foreignlanguage`).

Southeast Asian interword spacing
There is some preliminary interword spacing for Thai, Lao and Khemer in luatex (provided there are hyphenation patters) and xetex. It is activated automatically if a language with one of those scripts are loaded with `\babelprovide`. See the sample on the babel repository. With both engines, interword spacing is based on the "current" em unit (the size of the previous char in luatex and the font size set by the last `\selectfont` in xetex).

Bidi writing in luatex is still under development, but the basic implementation is finished. On the other hand, in xetex it is taking its first steps. The latter engine poses quite different challenges. An option to manage document layout in luatex (lists, footnotes, etc.) is almost finished, but xetex required more work.

Unfortunately, proper support for xetex requires patching somehow lots of macros and packages (and some issues related to `\specials` remain, like color and hyperlinks).

bidi=bidi
New 3.27 This package option is a new experimental support for bidi writing with xetex and the bidi package (by Vafa Khalighi). Currently, it just provides the basic direction switches with `\selectlanguage` and `\foreignlanguage`. Any help in making babel and bidi collaborate will be welcome (although the underlying concepts in both packages seem very different).

See the babel repository for a small example (`xe-bidi`).

Old stuff
A couple of tentative macros were provided by babel ($\geq$3.9g) with a partial solution for "Unicode" fonts. These macros are now deprecated — use `\babelfont`. A short description follows, for reference:

- `\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}
```

---

[23]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.

2   Loading languages with `language.dat`

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

New 3.9q   With luatex, however, patterns are loaded on the fly when requested by the language (except the "0th" language, typically english, which is preloaded always).[24] 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).[25]

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[26]. 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.[27] For example:

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

With the previous settings, if the encoding when the language is selected is T1 then the patterns in hyphenT1.ger are used, but otherwise use those in hyphen.ger (note the encoding 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
```

---

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

[25]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`.

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

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

```
rebuild the format. Now I will use the patterns
preloaded for english instead}}
```

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

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

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

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

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

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

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

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

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

Some recommendations:

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

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

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

- Do not switch scripts. If you want to make sure a set of glyphs is used, switch either the
  font encoding (low level) or the language (high level, which in turn may switch the font
  encoding). Usage of things like \latintext is deprecated.[28]

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

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

3.1 Guidelines for contributed languages

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

---

[28]But not removed, for backward compatibility.

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

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

	(Assigning \lefthyphenmin and \righthyphenmin directly in \extras<lang> has no effect.)
\providehyphenmins	The macro \providehyphenmins should be used in the language definition files to set \lefthyphenmin and \righthyphenmin. This macro will check whether these parameters were provided by the hyphenation file before it takes any action. If these values have been already set, this command is ignored (currently, default pattern files do *not* set them).
\captions⟨*lang*⟩	The macro \captions⟨*lang*⟩ defines the macros that hold the texts to replace the original hard-wired texts.
\date⟨*lang*⟩	The macro \date⟨*lang*⟩ defines \today.
\extras⟨*lang*⟩	The macro \extras⟨*lang*⟩ contains all the extra definitions needed for a specific language. This macro, like the following, is a hook – you can add things to it, but it must not be used directly.
\noextras⟨*lang*⟩	Because we want to let the user switch between languages, but we do not know what state TeX might be in after the execution of \extras⟨*lang*⟩, a macro that brings TeX into a predefined state is needed. It will be no surprise that the name of this macro is \noextras⟨*lang*⟩.
\bbl@declare@ttribute	This is a command to be used in the language definition files for declaring a language attribute. It takes three arguments: the name of the language, the attribute to be defined, and the code to be executed when the attribute is to be used.
\main@language	To postpone the activation of the definitions needed for a language until the beginning of a document, all language definition files should use \main@language instead of \selectlanguage. This will just store the name of the language, and the proper language will be activated at the start of the document.
\ProvidesLanguage	The macro \ProvidesLanguage should be used to identify the language definition files. Its syntax is similar to the syntax of the LaTeX command \ProvidesPackage.
\LdfInit	The macro \LdfInit performs a couple of standard checks that must be made at the beginning of a language definition file, such as checking the category code of the @-sign, preventing the .ldf file from being processed twice, etc.
\ldf@quit	The macro \ldf@quit does work needed if a .ldf file was processed earlier. This includes resetting the category code of the @-sign, preparing the language to be activated at \begin{document} time, and ending the input stream.
\ldf@finish	The macro \ldf@finish does work needed at the end of each .ldf file. This includes resetting the category code of the @-sign, loading a local configuration file, and preparing the language to be activated at \begin{document} time.
\loadlocalcfg	After processing a language definition file, LaTeX can be instructed to load a local configuration file. This file can, for instance, be used to add strings to \captions⟨*lang*⟩ to support local document classes. The user will be informed that this configuration file has been loaded. This macro is called by \ldf@finish.
\substitutefontfamily	(Deprecated.) This command takes three arguments, a font encoding and two font family names. It creates a font description file for the first font in the given encoding. This .fd file will instruct LaTeX to use a font from the second family when a font from the first family in the given encoding seems to be needed.

3.3 Skeleton

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

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

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

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

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

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

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

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

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

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

\EndBabelCommands

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

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

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

```
\AtEndOfPackage{%
  \RequirePackage{dingbat}%        Delay package
```

```
    \savebox{\myeye}{\eye}}%        And direct usage
  \newsavebox{\myeye}
  \newcommand\myanchor{\anchor}%    But OK inside command
```

3.4   Support for active characters

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

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

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

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

\bbl@add@special   The TeXbook states: "Plain TeX includes a macro called `\dospecials` that is essentially a set
\bbl@remove@special   macro, representing the set of all characters that have a special category code." [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[29].

\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`.

---

[29]This mechanism was introduced by Bernd Raichle.

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

3.8 Encoding-dependent strings

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

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

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

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

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

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

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

If a string is set several times (because several blocks are read), the first one 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 translated to the internal representation used by the engine, typically utf8, which is the only value supported currently (default is no translations). Note charset is applied by luatex and xetex when reading the file, not when the macro or string is used in the document.

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

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

The ⟨*category*⟩ is either `captions`, `date` or `extras`. You must stick to these three categories, even if no error is raised when using other name.[30] It may be empty, too, but in such a case using `\SetString` is an error (but not `\SetCase`).

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

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

\EndBabelCommands
```

A real example is:

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

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

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

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

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

---

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

```
    \SetString\today{\number\day.~%
      \csname month\romannumeral\month name\endcsname\space
      \number\year}

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

  \EndBabelCommands
```

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

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

The starred version just forces `strings` to take a value – if not set as package option, then the default for the engine is used. This is not done by default to prevent backward incompatibilities, but if you are creating a new language this version is better. It's up to the maintainers of the current languages to decide if using it is appropriate.[31]

\EndBabelCommands   Marks the end of the series of blocks.

\AfterBabelCommands   {⟨*code*⟩}

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

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

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

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

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

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

#1 is replaced by the roman numeral.

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

Sets globally code to be executed at \MakeUppercase and \MakeLowercase. The code would 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

---

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

arguments take precedence over the optional one. This command, unlike `\SetString`, is executed always (even without `strings`), and it is intended for minor readjustments only. For example, as T1 is the default case mapping in LaTeX, we 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 were related to bugs, either to fix them (there were lots), or to provide some alternatives. Even new features like \babelhyphen are intended to solve a certain problem (in this case, the lacking of a uniform syntax and behavior for shorthands across languages). These changes are described in this manual in the corresponding place. A selective list follows:

- \select@language did not set \languagename. This meant the language in force when auxiliary files were loaded was the one used in, for example, shorthands – if the language was german, a \select@language{spanish} had no effect.

- \foreignlanguage and otherlanguage* messed up \extras<language>. Scripts, encodings and many other things were not switched correctly.

- The :ENC mechanism for hyphenation patterns used the encoding of the *previous* language, not that of the language being selected.

- ' (with activeacute) had the original value when writing to an auxiliary file, and things like an infinite loop 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.

Part II
Source code

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

5 Identification and loading of required files

Code documentation is still under revision.
The 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 ⟨⟨*name*⟩⟩. That brings a little bit of literate programming.

6 `locale` **directory**

A required component of babel is a set of `ini` files with basic definitions for about 200 languages. They are distributed as a separate `zip` file, not packed as `dtx`. With them, babel will fully support Unicode engines.
Most of them are essentially finished (except bugs and mistakes, of course). Some of them are still incomplete (but they will be usable), and there are some omissions (eg, Latin and polytonic Greek, and there are no geographic areas in Spanish). Hindi, French, Occitan and Breton will show a warning related to dates. Not all include LICR variants.
This is a preliminary documentation.
`ini` files contain the actual data; `tex` files are currently just proxies to the corresponding `ini` files.
Most keys are self-explanatory.

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

Keys may be further qualified in a particular language with a suffix starting with a uppercase letter. It can be just a letter (eg, babel.name.A, babel.name.B) or a name (eg, date.long.Nominative, date.long.Formal, but no language is currently using the latter). Multi-letter qualifiers are forward compatible in the sense they won't conflict with new "global" keys (all lowercase).

7 **Tools**

```
1 ⟨⟨version=3.29⟩⟩
2 ⟨⟨date=2019/04/03⟩⟩
```

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[32]. These
macros will break if another \if...\fi statement appears in one of the arguments and it
is not enclosed in braces.

```
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 ϵ-tex engine, it is based on \ifcsname, which is more
efficient, and do not waste memory.

```
41 \def\bbl@ifunset#1{%
```

---

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

```
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}%
```

53

```
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⟩⟩
```

7.1   Multiple languages

\language   Plain TeX version 3.0 provides the primitive \language that is used to store the current language. When used with a pre-3.0 version this function has to be implemented by allocating a counter. The following block is used in `switch.def` and `hyphen.cfg`; the latter may seem redundant, but remember babel doesn't requires loading `switch.def` in the format.

```
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.

8 The Package File (LaTeX, `babel.sty`)

In order to make use of the features of LaTeX 2_ε, 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.

8.1 `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    \let\bbl@tempa\relax
199    \@ifpackagewith{babel}{bidi=basic}%
200      {\def\bbl@tempa{basic}}%
201      {\@ifpackagewith{babel}{bidi=basic-r}%
202        {\def\bbl@tempa{basic-r}}%
203        {}}
204    \ifx\bbl@tempa\relax\else
205      \let\bbl@beforeforeign\leavevmode
206      \AtEndOfPackage{\EnableBabelHook{babel-bidi}}%
207      \RequirePackage{luatexbase}%
208      \directlua{
209        require('babel-bidi.lua')
210        require('babel-bidi-\bbl@tempa.lua')
211        luatexbase.add_to_callback('pre_linebreak_filter',
212          Babel.pre_otfload_v,
213          'Babel.pre_otfload_v',
214          luatexbase.priority_in_callback('pre_linebreak_filter',
215            'luaotfload.node_processor') or nil)
216        luatexbase.add_to_callback('hpack_filter',
217          Babel.pre_otfload_h,
218          'Babel.pre_otfload_h',
219          luatexbase.priority_in_callback('hpack_filter',
220            'luaotfload.node_processor') or nil)
221      }
222    \fi
223 \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.

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

8.2 `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.

```
239 \bbl@trace{key=value and another general options}
240 \bbl@csarg\let{tempa\expandafter}\csname opt@babel.sty\endcsname
241 \def\bbl@tempb#1.#2{%
242    #1\ifx\@empty#2\else,\bbl@afterfi\bbl@tempb#2\fi}%
243 \def\bbl@tempd#1.#2\@nnil{%
244   \ifx\@empty#2%
245     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
246   \else
247     \in@{=}{#1}\ifin@
248       \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.#2}%
249     \else
250       \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
251       \bbl@csarg\edef{mod@#1}{\bbl@tempb#2}%
252     \fi
253   \fi}
254 \let\bbl@tempc\@empty
255 \bbl@foreach\bbl@tempa{\bbl@tempd#1.\@empty\@nnil}
256 \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.

```
257 \DeclareOption{KeepShorthandsActive}{}
258 \DeclareOption{activeacute}{}
259 \DeclareOption{activegrave}{}
260 \DeclareOption{debug}{}
261 \DeclareOption{noconfigs}{}
262 \DeclareOption{showlanguages}{}
263 \DeclareOption{silent}{}
264 \DeclareOption{mono}{}
265 \DeclareOption{shorthands=off}{\bbl@tempa shorthands=\bbl@tempa}
266 ⟨⟨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.

```
267 \let\bbl@opt@shorthands\@nnil
268 \let\bbl@opt@config\@nnil
269 \let\bbl@opt@main\@nnil
270 \let\bbl@opt@headfoot\@nnil
271 \let\bbl@opt@layout\@nnil
```

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

```
272 \def\bbl@tempa#1=#2\bbl@tempa{%
273   \bbl@csarg\ifx{opt@#1}\@nnil
274     \bbl@csarg\edef{opt@#1}{#2}%
275   \else
276     \bbl@error{%
```

58

```
277        Bad option `#1=#2'. Either you have misspelled the\\%
278        key or there is a previous setting of `#1'}{%
279        Valid keys are `shorthands', `config', `strings', `main',\\%
280        `headfoot', `safe', `math', among others.}
281    \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.

```
282 \let\bbl@language@opts\@empty
283 \DeclareOption*{%
284    \bbl@xin@{\string=}{\CurrentOption}%
285    \ifin@
286      \expandafter\bbl@tempa\CurrentOption\bbl@tempa
287    \else
288      \bbl@add@list\bbl@language@opts{\CurrentOption}%
289    \fi}
```

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

```
290 \ProcessOptions*
```

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

```
291 \bbl@trace{Conditional loading of shorthands}
292 \def\bbl@sh@string#1{%
293    \ifx#1\@empty\else
294      \ifx#1t\string~%
295      \else\ifx#1c\string,%
296      \else\string#1%
297      \fi\fi
298      \expandafter\bbl@sh@string
299    \fi}
300 \ifx\bbl@opt@shorthands\@nnil
301    \def\bbl@ifshorthand#1#2#3{#2}%
302 \else\ifx\bbl@opt@shorthands\@empty
303    \def\bbl@ifshorthand#1#2#3{#3}%
304 \else
```

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

```
305    \def\bbl@ifshorthand#1{%
306      \bbl@xin@{\string#1}{\bbl@opt@shorthands}%
307      \ifin@
308        \expandafter\@firstoftwo
309      \else
310        \expandafter\@secondoftwo
311      \fi}
```

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

```
312    \edef\bbl@opt@shorthands{%
313      \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.

```
314 \bbl@ifshorthand{'}%
315    {\PassOptionsToPackage{activeacute}{babel}}{}
316 \bbl@ifshorthand{`}%
317    {\PassOptionsToPackage{activegrave}{babel}}{}
318 \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.

```
319 \ifx\bbl@opt@headfoot\@nnil\else
320   \g@addto@macro\@resetactivechars{%
321     \set@typeset@protect
322     \expandafter\select@language@x\expandafter{\bbl@opt@headfoot}%
323     \let\protect\noexpand}
324 \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.

```
325 \ifx\bbl@opt@safe\@undefined
326   \def\bbl@opt@safe{BR}
327 \fi
328 \ifx\bbl@opt@main\@nnil\else
329   \edef\bbl@language@opts{%
330     \ifx\bbl@language@opts\@empty\else\bbl@language@opts,\fi
331       \bbl@opt@main}
332 \fi
```

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

```
333 \bbl@trace{Defining IfBabelLayout}
334 \ifx\bbl@opt@layout\@nnil
335   \newcommand\IfBabelLayout[3]{#3}%
336 \else
337   \newcommand\IfBabelLayout[1]{%
338     \@expandtwoargs\in@{.#1.}{.\bbl@opt@layout.}%
339     \ifin@
340       \expandafter\@firstoftwo
341     \else
342       \expandafter\@secondoftwo
343     \fi}
344 \fi
```

8.4   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).

```
345 \bbl@trace{Language options}
346 \let\bbl@afterlang\relax
347 \let\BabelModifiers\relax
348 \let\bbl@loaded\@empty
349 \def\bbl@load@language#1{%
350   \InputIfFileExists{#1.ldf}%
351     {\edef\bbl@loaded{\CurrentOption
352       \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}%
```

```
353     \expandafter\let\expandafter\bbl@afterlang
354       \csname\CurrentOption.ldf-h@@k\endcsname
355     \expandafter\let\expandafter\BabelModifiers
356       \csname bbl@mod@\CurrentOption\endcsname}%
357    {\bbl@error{%
358      Unknown option `\CurrentOption'. Either you misspelled it\\%
359      or the language definition file \CurrentOption.ldf was not found}{%
360      Valid options are: shorthands=, KeepShorthandsActive,\\%
361      activeacute, activegrave, noconfigs, safe=, main=, math=\\%
362      headfoot=, strings=, config=, hyphenmap=, or a language name.}}}
```

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

```
363 \def\bbl@try@load@lang#1#2#3{%
364    \IfFileExists{\CurrentOption.ldf}%
365      {\bbl@load@language{\CurrentOption}}%
366      {#1\bbl@load@language{#2}#3}}
367 \DeclareOption{afrikaans}{\bbl@try@load@lang{}{dutch}{}}
368 \DeclareOption{brazil}{\bbl@try@load@lang{}{portuges}{}}
369 \DeclareOption{brazilian}{\bbl@try@load@lang{}{portuges}{}}
370 \DeclareOption{hebrew}{%
371   \input{rlbabel.def}%
372   \bbl@load@language{hebrew}}
373 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}{}}
374 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}{}}
375 \DeclareOption{nynorsk}{\bbl@try@load@lang{}{norsk}{}}
376 \DeclareOption{polutonikogreek}{%
377   \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}
378 \DeclareOption{portuguese}{\bbl@try@load@lang{}{portuges}{}}
379 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}{}}
380 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}{}}
381 \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.

```
382 \ifx\bbl@opt@config\@nnil
383   \@ifpackagewith{babel}{noconfigs}{}%
384     {\InputIfFileExists{bblopts.cfg}%
385       {\typeout{*************************************^^J%
386                * Local config file bblopts.cfg used^^J%
387                *}}%
388     {}}%
389 \else
390   \InputIfFileExists{\bbl@opt@config.cfg}%
391     {\typeout{*************************************^^J%
392               * Local config file \bbl@opt@config.cfg used^^J%
393               *}}%
394     {\bbl@error{%
395        Local config file `\bbl@opt@config.cfg' not found}{%
396        Perhaps you misspelled it.}}%
397 \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.

```
398 \bbl@for\bbl@tempa\bbl@language@opts{%
399   \bbl@ifunset{ds@\bbl@tempa}%
400     {\edef\bbl@tempb{%
401        \noexpand\DeclareOption
402          {\bbl@tempa}%
403          {\noexpand\bbl@load@language{\bbl@tempa}}}%
404      \bbl@tempb}%
405      \@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.

```
406 \bbl@foreach\@classoptionslist{%
407   \bbl@ifunset{ds@#1}%
408     {\IfFileExists{#1.ldf}%
409        {\DeclareOption{#1}{\bbl@load@language{#1}}}%
410        {}}%
411     {}}
```

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

```
412 \ifx\bbl@opt@main\@nnil\else
413   \expandafter
414   \let\expandafter\bbl@loadmain\csname ds@\bbl@opt@main\endcsname
415   \DeclareOption{\bbl@opt@main}{}
416 \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):

```
417 \def\AfterBabelLanguage#1{%
418   \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang}{}}
419 \DeclareOption*{}
420 \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.

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

```
439  \ExecuteOptions{\bbl@opt@main}
440  \DeclareOption*{}
441  \ProcessOptions*
442 \fi
443 \def\AfterBabelLanguage{%
444   \bbl@error
445     {Too late for \string\AfterBabelLanguage}%
446     {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.

```
447 \ifx\bbl@main@language\@undefined
448   \bbl@info{%
449     You haven't specified a language. I'll use 'nil'\\%
450     as the main language. Reported}
451     \bbl@load@language{nil}
452 \fi
453 ⟨/package⟩
454 ⟨*core⟩
```

9   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.

9.1   Tools

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

The file `babel.def` expects some definitions made in the LaTeX 2$_\varepsilon$ 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).

```
462 \ifx\bbl@ifshorthand\@undefined
463   \let\bbl@opt@shorthands\@nnil
```

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

And continue.

```
483 \ifx\bbl@switchflag\@undefined % Prevent double input
484   \let\bbl@switchflag\relax
485   \input switch.def\relax
486 \fi
487 \bbl@trace{Compatibility with language.def}
488 \ifx\bbl@languages\@undefined
489   \ifx\directlua\@undefined
490     \openin1 = language.def
491     \ifeof1
492       \closein1
493       \message{I couldn't find the file language.def}
494     \else
495       \closein1
496       \begingroup
497         \def\addlanguage#1#2#3#4#5{%
498           \expandafter\ifx\csname lang@#1\endcsname\relax\else
499             \global\expandafter\let\csname l@#1\expandafter\endcsname
500               \csname lang@#1\endcsname
501           \fi}%
502         \def\uselanguage#1{}%
503         \input language.def
504       \endgroup
505     \fi
506   \fi
507   \chardef\l@english\z@
508 \fi
509 ⟨⟨Load patterns in luatex⟩⟩
510 ⟨⟨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 *re*defined, using the contents of the token register.

```
511 \def\addto#1#2{%
512   \ifx#1\@undefined
513     \def#1{#2}%
514   \else
515     \ifx#1\relax
516       \def#1{#2}%
517     \else
518       {\toks@\expandafter{#1#2}%
519         \xdef#1{\the\toks@}}%
520     \fi
521   \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.

```
522 \def\bbl@withactive#1#2{%
523   \begingroup
524     \lccode`~=`#2\relax
525     \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@...

```
526 \def\bbl@redefine#1{%
527   \edef\bbl@tempa{\bbl@stripslash#1}%
528   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
529   \expandafter\def\csname\bbl@tempa\endcsname}
```

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

```
530 \@onlypreamble\bbl@redefine
```

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

```
531 \def\bbl@redefine@long#1{%
532   \edef\bbl@tempa{\bbl@stripslash#1}%
533   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
534   \expandafter\long\expandafter\def\csname\bbl@tempa\endcsname}
535 \@onlypreamble\bbl@redefine@long
```

\bbl@redefinerobust   For commands that are redefined, but which *might* be robust we need a slightly more intelligent macro. A robust command foo is defined to expand to \protect\foo . So it is necessary to check whether \foo exists. The result is that the command that is being redefined is always robust afterwards. Therefore all we need to do now is define \foo .

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

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

```
543 \@onlypreamble\bbl@redefinerobust
```

9.2 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.

```
544 \bbl@trace{Hooks}
545 \def\AddBabelHook#1#2{%
546   \bbl@ifunset{bbl@hk@#1}{\EnableBabelHook{#1}}{}%
547   \def\bbl@tempa##1,#2=##2,##3\@empty{\def\bbl@tempb{##2}}%
548   \expandafter\bbl@tempa\bbl@evargs,#2=,\@empty
549   \bbl@ifunset{bbl@ev@#1@#2}%
550     {\bbl@csarg\bbl@add{ev@#2}{\bbl@elt{#1}}%
551      \bbl@csarg\newcommand}%
552     {\bbl@csarg\let{ev@#1@#2}\relax
553      \bbl@csarg\newcommand}%
554   {ev@#1@#2}[\bbl@tempb]}
555 \def\EnableBabelHook#1{\bbl@csarg\let{hk@#1}\@firstofone}
556 \def\DisableBabelHook#1{\bbl@csarg\let{hk@#1}\@gobble}
557 \def\bbl@usehooks#1#2{%
558   \def\bbl@elt##1{%
559     \@nameuse{bbl@hk@##1}{\@nameuse{bbl@ev@##1@#1}#2}}%
560   \@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).

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

\babelensure The user command just parses the optional argument and creates a new macro named \bbl@e@⟨language⟩. We register a hook at the afterextras event which just executes this macro in a "complete" selection (which, if undefined, is \relax and does nothing). This part is somewhat involved because we have to make sure things are expanded the correct number of times.

The macro \bbl@e@⟨language⟩ contains \bbl@ensure{⟨include⟩}{⟨exclude⟩}{⟨fontenc⟩}, which in in turn loops over the macros names in \bbl@captionslist, excluding (with the help of \in@) those in the exclude list. If the fontenc is given (and not \relax), the \fontencoding is also added. Then we loop over the include list, but if the macro already contains \foreignlanguage, nothing is done. Note this macro (1) is not restricted to the preamble, and (2) changes are local.

```
566 \bbl@trace{Defining babelensure}
567 \newcommand\babelensure[2][]{%  TODO - revise test files
568   \AddBabelHook{babel-ensure}{afterextras}{%
569     \ifcase\bbl@select@type
570       \@nameuse{bbl@e@\languagename}%
571     \fi}%
572   \begingroup
573     \let\bbl@ens@include\@empty
574     \let\bbl@ens@exclude\@empty
575     \def\bbl@ens@fontenc{\relax}%
576     \def\bbl@tempb##1{%
577       \ifx\@empty##1\else\noexpand##1\expandafter\bbl@tempb\fi}%
```

```
578    \edef\bbl@tempa{\bbl@tempb#1\@empty}%
579    \def\bbl@tempb##1=##2\@@{\@namedef{bbl@ens@##1}{##2}}%
580    \bbl@foreach\bbl@tempa{\bbl@tempb##1\@@}%
581    \def\bbl@tempc{\bbl@ensure}%
582    \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
583      \expandafter{\bbl@ens@include}}%
584    \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
585      \expandafter{\bbl@ens@exclude}}%
586    \toks@\expandafter{\bbl@tempc}%
587    \bbl@exp{%
588  \endgroup
589  \def\<bbl@e@#2>{\the\toks@{\bbl@ens@fontenc}}}}
590 \def\bbl@ensure#1#2#3{% 1: include 2: exclude 3: fontenc
591    \def\bbl@tempb##1{% elt for (excluding) \bbl@captionslist list
592    \ifx##1\@empty\else
593      \in@{##1}{#2}%
594      \ifin@\else
595        \bbl@ifunset{bbl@ensure@\languagename}%
596          {\bbl@exp{%
597            \\\DeclareRobustCommand\<bbl@ensure@\languagename>[1]{%
598              \\\foreignlanguage{\languagename}%
599              {\ifx\relax#3\else
600                \\\fontencoding{#3}\\\selectfont
601              \fi
602              ########1}}}}%
603          {}%
604        \toks@\expandafter{##1}%
605        \edef##1{%
606          \bbl@csarg\noexpand{ensure@\languagename}%
607          {\the\toks@}}%
608      \fi
609      \expandafter\bbl@tempb
610    \fi}%
611  \expandafter\bbl@tempb\bbl@captionslist\today\@empty
612  \def\bbl@tempa##1{% elt for include list
613    \ifx##1\@empty\else
614      \bbl@csarg\in@{ensure@\languagename\expandafter}\expandafter{##1}%
615      \ifin@\else
616        \bbl@tempb##1\@empty
617      \fi
618      \expandafter\bbl@tempa
619    \fi}%
620  \bbl@tempa#1\@empty}
621 \def\bbl@captionslist{%
622  \prefacename\refname\abstractname\bibname\chaptername\appendixname
623  \contentsname\listfigurename\listtablename\indexname\figurename
624  \tablename\partname\enclname\ccname\headtoname\pagename\seename
625  \alsoname\proofname\glossaryname}
```

9.3 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.

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

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

```
653 \def\ldf@quit#1{%
654   \expandafter\main@language\expandafter{#1}%
655   \catcode`\@=\atcatcode \let\atcatcode\relax
656   \catcode`\==\eqcatcode \let\eqcatcode\relax
657   \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.

```
658 \def\bbl@afterldf#1{%
659   \bbl@afterlang
660   \let\bbl@afterlang\relax
```

68

```
661    \let\BabelModifiers\relax
662    \let\bbl@screset\relax}%
663 \def\ldf@finish#1{%
664    \loadlocalcfg{#1}%
665    \bbl@afterldf{#1}%
666    \expandafter\main@language\expandafter{#1}%
667    \catcode`\@=\atcatcode \let\atcatcode\relax
668    \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.

```
669 \@onlypreamble\LdfInit
670 \@onlypreamble\ldf@quit
671 \@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.

```
672 \def\main@language#1{%
673    \def\bbl@main@language{#1}%
674    \let\languagename\bbl@main@language
675    \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.

```
676 \AtBeginDocument{%
677    \expandafter\selectlanguage\expandafter{\bbl@main@language}%
678    \ifcase\bbl@engine\or\pagedir\bodydir\fi}  % TODO - a better place
```

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

```
679 \def\select@language@x#1{%
680    \ifcase\bbl@select@type
681      \bbl@ifsamestring\languagename{#1}{}{\select@language{#1}}%
682    \else
683      \select@language{#1}%
684    \fi}
```

9.4 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.

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

69

```
694        \endgroup
695        \bbl@add\nfss@catcodes{\@makeother#1}%
696      \else
697        \endgroup
698      \fi
699    \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.

```
700 \def\bbl@remove@special#1{%
701   \begingroup
702     \def\x##1##2{\ifnum`#1=`##2\noexpand\@empty
703                 \else\noexpand##1\noexpand##2\fi}%
704     \def\do{\x\do}%
705     \def\@makeother{\x\@makeother}%
706   \edef\x{\endgroup
707     \def\noexpand\dospecials{\dospecials}%
708     \expandafter\ifx\csname @sanitize\endcsname\relax\else
709       \def\noexpand\@sanitize{\@sanitize}%
710     \fi}%
711   \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).

```
712 \def\bbl@active@def#1#2#3#4{%
713   \@namedef{#3#1}{%
714     \expandafter\ifx\csname#2@sh@#1@\endcsname\relax
715       \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
716     \else
717       \bbl@afterfi\csname#2@sh@#1@\endcsname
718     \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.

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

```
723        \bbl@afterfi\csname#2@sh@#1@\string##1@\endcsname
724      \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.

```
725 \def\initiate@active@char#1{%
726   \bbl@ifunset{active@char\string#1}%
727     {\bbl@withactive
728        {\expandafter\@initiate@active@char\expandafter}#1\string#1#1}%
729     {}}
```

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`).

```
730 \def\@initiate@active@char#1#2#3{%
731   \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
732   \ifx#1\@undefined
733     \bbl@csarg\edef{oridef@#2}{\let\noexpand#1\noexpand\@undefined}%
734   \else
735     \bbl@csarg\let{oridef@@#2}#1%
736     \bbl@csarg\edef{oridef@#2}{%
737       \let\noexpand#1%
738       \expandafter\noexpand\csname bbl@oridef@@#2\endcsname}%
739   \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*).

```
740   \ifx#1#3\relax
741     \expandafter\let\csname normal@char#2\endcsname#3%
742   \else
743     \bbl@info{Making #2 an active character}%
744     \ifnum\mathcode`#2="8000
745       \@namedef{normal@char#2}{%
746         \textormath{#3}{\csname bbl@oridef@@#2\endcsname}}%
747     \else
748       \@namedef{normal@char#2}{#3}%
749     \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).

```
750   \bbl@restoreactive{#2}%
751   \AtBeginDocument{%
752     \catcode`#2\active
753     \if@filesw
754       \immediate\write\@mainaux{\catcode`\string#2\active}%
755     \fi}%
756   \expandafter\bbl@add@special\csname#2\endcsname
757   \catcode`#2\active
```

```
758    \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*⟩).

```
759    \let\bbl@tempa\@firstoftwo
760    \if\string^#2%
761      \def\bbl@tempa{\noexpand\textormath}%
762    \else
763      \ifx\bbl@mathnormal\@undefined\else
764        \let\bbl@tempa\bbl@mathnormal
765      \fi
766    \fi
767    \expandafter\edef\csname active@char#2\endcsname{%
768      \bbl@tempa
769        {\noexpand\if@safe@actives
770          \noexpand\expandafter
771          \expandafter\noexpand\csname normal@char#2\endcsname
772        \noexpand\else
773          \noexpand\expandafter
774          \expandafter\noexpand\csname bbl@doactive#2\endcsname
775        \noexpand\fi}%
776      {\expandafter\noexpand\csname normal@char#2\endcsname}}%
777    \bbl@csarg\edef{doactive#2}{%
778      \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!).

```
779    \bbl@csarg\edef{active@#2}{%
780      \noexpand\active@prefix\noexpand#1%
781      \expandafter\noexpand\csname active@char#2\endcsname}%
782    \bbl@csarg\edef{normal@#2}{%
783      \noexpand\active@prefix\noexpand#1%
784      \expandafter\noexpand\csname normal@char#2\endcsname}%
785    \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.

```
786    \bbl@active@def#2\user@group{user@active}{language@active}%
787    \bbl@active@def#2\language@group{language@active}{system@active}%
788    \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.

```
789    \expandafter\edef\csname\user@group @sh@#2@@\endcsname
790      {\expandafter\noexpand\csname normal@char#2\endcsname}%
791    \expandafter\edef\csname\user@group @sh@#2@\string\protect@\endcsname
792      {\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.

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

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

```
798 ⟨⟨∗More package options⟩⟩ ≡
799 \DeclareOption{math=active}{}
800 \DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
801 ⟨⟨/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.

```
802 \@ifpackagewith{babel}{KeepShorthandsActive}%
803   {\let\bbl@restoreactive\@gobble}%
804   {\def\bbl@restoreactive#1{%
805     \bbl@exp{%
806       \\\AfterBabelLanguage\\\CurrentOption
807         {\catcode`#1=\the\catcode`#1\relax}%
808       \\\AtEndOfPackage
809         {\catcode`#1=\the\catcode`#1\relax}}}%
810   \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.

```
811 \def\bbl@sh@select#1#2{%
812   \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
813     \bbl@afterelse\bbl@scndcs
814   \else
815     \bbl@afterfi\csname#1@sh@#2@sel\endcsname
816   \fi}
```

\active@prefix   The command \active@prefix which is used in the expansion of active characters has a function similar to \OT1-cmd in that it \protects the active character whenever \protect is *not* \@typeset@protect.

```
817 \def\active@prefix#1{%
818   \ifx\protect\@typeset@protect
819   \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).

```
820       \ifx\protect\@unexpandable@protect
821         \noexpand#1%
```

73

```
822      \else
823        \protect#1%
824      \fi
825      \expandafter\@gobble
826    \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⟩.

```
827 \newif\if@safe@actives
828 \@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.

```
829 \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.

```
830 \def\bbl@activate#1{%
831    \bbl@withactive{\expandafter\let\expandafter}#1%
832      \csname bbl@active@\string#1\endcsname}
833 \def\bbl@deactivate#1{%
834    \bbl@withactive{\expandafter\let\expandafter}#1%
835      \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.

```
836 \def\bbl@firstcs#1#2{\csname#1\endcsname}
837 \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.

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

```
855        {\def\bbl@tempa{#4}%
856         \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
857         \else
858           \bbl@info
859             {Redefining #1 shorthand \string#2\string#3\\%
860              in language \CurrentOption}%
861         \fi}%
862      \@namedef{#1@sh@\string#2@\string#3@}{#4}%
863    \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.

```
864 \def\textormath{%
865   \ifmmode
866     \expandafter\@secondoftwo
867   \else
868     \expandafter\@firstoftwo
869   \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'.

```
870 \def\user@group{user}
871 \def\language@group{english}
872 \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.

```
873 \def\useshorthands{%
874   \@ifstar\bbl@usesh@s{\bbl@usesh@x{}}}
875 \def\bbl@usesh@s#1{%
876   \bbl@usesh@x
877     {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bbl@activate{#1}}}%
878     {#1}}
879 \def\bbl@usesh@x#1#2{%
880   \bbl@ifshorthand{#2}%
881     {\def\user@group{user}%
882      \initiate@active@char{#2}%
883      #1%
884      \bbl@activate{#2}}%
885     {\bbl@error
886        {Cannot declare a shorthand turned off (\string#2)}
887        {Sorry, but you cannot use shorthands which have been\\%
888         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.

```
889 \def\user@language@group{user@\language@group}
890 \def\bbl@set@user@generic#1#2{%
891   \bbl@ifunset{user@generic@active#1}%
```

```
892        {\bbl@active@def#1\user@language@group{user@active}{user@generic@active}%
893         \bbl@active@def#1\user@group{user@generic@active}{language@active}%
894         \expandafter\edef\csname#2@sh@#1@@\endcsname{%
895           \expandafter\noexpand\csname normal@char#1\endcsname}%
896         \expandafter\edef\csname#2@sh@#1@\string\protect@\endcsname{%
897           \expandafter\noexpand\csname user@active#1\endcsname}}%
898     \@empty}
899 \newcommand\defineshorthand[3][user]{%
900     \edef\bbl@tempa{\zap@space#1 \@empty}%
901     \bbl@for\bbl@tempb\bbl@tempa{%
902       \if*\expandafter\@car\bbl@tempb\@nil
903         \edef\bbl@tempb{user@\expandafter\@gobble\bbl@tempb}%
904         \@expandtwoargs
905           \bbl@set@user@generic{\expandafter\string\@car#2\@nil}\bbl@tempb
906       \fi
907       \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.

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

\aliasshorthand   First the new shorthand needs to be initialized,

```
909 \def\aliasshorthand#1#2{%
910     \bbl@ifshorthand{#2}%
911       {\expandafter\ifx\csname active@char\string#2\endcsname\relax
912         \ifx\document\@notprerr
913           \@notshorthand{#2}%
914         \else
915           \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".

```
916           \expandafter\let\csname active@char\string#2\expandafter\endcsname
917             \csname active@char\string#1\endcsname
918           \expandafter\let\csname normal@char\string#2\expandafter\endcsname
919             \csname normal@char\string#1\endcsname
920           \bbl@activate{#2}%
921         \fi
922       \fi}%
923       {\bbl@error
924         {Cannot declare a shorthand turned off (\string#2)}
925         {Sorry, but you cannot use shorthands which have been\\%
926          turned off in the package options}}}
```

\@notshorthand

```
927 \def\@notshorthand#1{%
928     \bbl@error{%
929       The character `\string #1' should be made a shorthand character;\\%
930       add the command \string\useshorthands\string{#1\string} to
931       the preamble.\\%
932       I will ignore your instruction}%
933     {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.

```
934 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
935 \DeclareRobustCommand*\shorthandoff{%
936   \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
937 \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.

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

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

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

```
978      \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
979 \fi
```

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

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

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

```
981 \def\bbl@prim@s{%
982   \prime\futurelet\@let@token\bbl@pr@m@s}
983 \def\bbl@if@primes#1#2{%
984   \ifx#1\@let@token
985     \expandafter\@firstoftwo
986   \else\ifx#2\@let@token
987     \bbl@afterelse\expandafter\@firstoftwo
988   \else
989     \bbl@afterfi\expandafter\@secondoftwo
990   \fi\fi}
991 \begingroup
992   \catcode`\^=7  \catcode`\*=\active  \lccode`\*=`\^
993   \catcode`\'=12 \catcode`\"=\active  \lccode`\"=`\'
994   \lowercase{%
995     \gdef\bbl@pr@m@s{%
996       \bbl@if@primes"'%
997         \pr@@@s
998         {\bbl@if@primes*^\pr@@@t\egroup}}}
999 \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).

```
1000 \initiate@active@char{~}
1001 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
1002 \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.

```
1003 \expandafter\def\csname OT1dqpos\endcsname{127}
1004 \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

```
1005 \ifx\f@encoding\@undefined
1006   \def\f@encoding{OT1}
1007 \fi
```

9.5 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.

```
1008 \bbl@trace{Language attributes}
1009 \newcommand\languageattribute[2]{%
1010    \def\bbl@tempc{#1}%
1011    \bbl@fixname\bbl@tempc
1012    \bbl@iflanguage\bbl@tempc{%
1013       \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.

```
1014          \ifx\bbl@known@attribs\@undefined
1015             \in@false
1016          \else
```

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

```
1017             \bbl@xin@{,\bbl@tempc-##1,}{,\bbl@known@attribs,}%
1018          \fi
```

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

```
1019          \ifin@
1020             \bbl@warning{%
1021                You have more than once selected the attribute '##1'\\%
1022                for language #1. Reported}%
1023          \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.

```
1024             \bbl@exp{%
1025                \\\bbl@add@list\\\bbl@known@attribs{\bbl@tempc-##1}}%
1026             \edef\bbl@tempa{\bbl@tempc-##1}%
1027             \expandafter\bbl@ifknown@ttrib\expandafter{\bbl@tempa}\bbl@attributes%
1028             {\csname\bbl@tempc @attr@##1\endcsname}%
1029             {\@attrerr{\bbl@tempc}{##1}}%
1030          \fi}}}
```

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

```
1031 \@onlypreamble\languageattribute
```

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

```
1032 \newcommand*{\@attrerr}[2]{%
1033    \bbl@error
1034       {The attribute #2 is unknown for language #1.}%
1035       {Your command will be ignored, type <return> to proceed}}
```

\bbl@declare@ttribute   This command adds the new language/attribute combination to the list of known
                         attributes.
                         Then it defines a control sequence to be executed when the attribute is used in a
                         document. The result of this should be that the macro \extras... for the current
                         language is extended, otherwise the attribute will not work as its code is removed from
                         memory at \begin{document}.

```
1036 \def\bbl@declare@ttribute#1#2#3{%
1037    \bbl@xin@{,#2,}{,\BabelModifiers,}%
1038    \ifin@
1039       \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
```

79

```
1040    \fi
1041    \bbl@add@list\bbl@attributes{#1-#2}%
1042    \expandafter\def\csname#1@attr@#2\endcsname{#3}}
```

\bbl@ifattributeset This internal macro has 4 arguments. It can be used to interpret TeX code based on whether a certain attribute was set. This command should appear inside the argument to \AtBeginDocument because the attributes are set in the document preamble, *after* babel is loaded.
The first argument is the language, the second argument the attribute being checked, and the third and fourth arguments are the true and false clauses.

```
1043 \def\bbl@ifattributeset#1#2#3#4{%
```

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

```
1044    \ifx\bbl@known@attribs\@undefined
1045      \in@false
1046    \else
```

The we need to check the list of known attributes.

```
1047      \bbl@xin@{,#1-#2,}{,\bbl@known@attribs,}%
1048    \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'.

```
1049    \ifin@
1050      \bbl@afterelse#3%
1051    \else
1052      \bbl@afterfi#4%
1053    \fi
1054    }
```

\bbl@ifknown@ttrib An internal macro to check whether a given language/attribute is known. The macro takes 4 arguments, the language/attribute, the attribute list, the TeX-code to be executed when the attribute is known and the TeX-code to be executed otherwise.

```
1055 \def\bbl@ifknown@ttrib#1#2{%
```

We first assume the attribute is unknown.

```
1056    \let\bbl@tempa\@secondoftwo
```

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

```
1057    \bbl@loopx\bbl@tempb{#2}{%
1058      \expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{,#1,}%
1059      \ifin@
```

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

```
1060        \let\bbl@tempa\@firstoftwo
1061      \else
1062      \fi}%
```

Finally we execute \bbl@tempa.

```
1063    \bbl@tempa
1064 }
```

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

```
1065 \def\bbl@clear@ttribs{%
1066    \ifx\bbl@attributes\@undefined\else
1067      \bbl@loopx\bbl@tempa{\bbl@attributes}{%
```

```
1068        \expandafter\bbl@clear@ttrib\bbl@tempa.
1069        }%
1070    \let\bbl@attributes\@undefined
1071  \fi}
1072 \def\bbl@clear@ttrib#1-#2.{%
1073   \expandafter\let\csname#1@attr@#2\endcsname\@undefined}
1074 \AtBeginDocument{\bbl@clear@ttribs}
```

9.6 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
```
1075 \bbl@trace{Macros for saving definitions}
1076 \def\babel@beginsave{\babel@savecnt\z@}
```

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

```
1077 \newcount\babel@savecnt
1078 \babel@beginsave
```

\babel@save   The macro \babel@save⟨csname⟩ saves the current meaning of the control sequence
⟨csname⟩ to \originalTeX[33]. 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.

```
1079 \def\babel@save#1{%
1080   \expandafter\let\csname babel@\number\babel@savecnt\endcsname#1\relax
1081   \toks@\expandafter{\originalTeX\let#1=}%
1082   \bbl@exp{%
1083     \def\\\originalTeX{\the\toks@\<babel@\number\babel@savecnt>\relax}}%
1084   \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.

```
1085 \def\babel@savevariable#1{%
1086   \toks@\expandafter{\originalTeX #1=}%
1087   \bbl@exp{\def\\\originalTeX{\the\toks@\the#1\relax}}}
```

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

```
1088 \def\bbl@frenchspacing{%
1089   \ifnum\the\sfcode`\.=\@m
1090     \let\bbl@nonfrenchspacing\relax
1091   \else
1092     \frenchspacing
1093     \let\bbl@nonfrenchspacing\nonfrenchspacing
1094   \fi}
1095 \let\bbl@nonfrenchspacing\nonfrenchspacing
```

---

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

9.7 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.

```
1096 \bbl@trace{Short tags}
1097 \def\babeltags#1{%
1098   \edef\bbl@tempa{\zap@space#1 \@empty}%
1099   \def\bbl@tempb##1=##2\@@{%
1100     \edef\bbl@tempc{%
1101       \noexpand\newcommand
1102       \expandafter\noexpand\csname ##1\endcsname{%
1103         \noexpand\protect
1104         \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}%
1105       \noexpand\newcommand
1106       \expandafter\noexpand\csname text##1\endcsname{%
1107         \noexpand\foreignlanguage{##2}}}%
1108     \bbl@tempc}%
1109   \bbl@for\bbl@tempa\bbl@tempa{%
1110     \expandafter\bbl@tempb\bbl@tempa\@@}}
```

9.8 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.

```
1111 \bbl@trace{Hyphens}
1112 \@onlypreamble\babelhyphenation
1113 \AtEndOfPackage{%
1114   \newcommand\babelhyphenation[2][\@empty]{%
1115     \ifx\bbl@hyphenation@\relax
1116       \let\bbl@hyphenation@\@empty
1117     \fi
1118     \ifx\bbl@hyphlist\@empty\else
1119       \bbl@warning{%
1120         You must not intermingle \string\selectlanguage\space and\\%
1121         \string\babelhyphenation\space or some exceptions will not\\%
1122         be taken into account. Reported}%
1123     \fi
1124     \ifx\@empty#1%
1125       \protected@edef\bbl@hyphenation@{\bbl@hyphenation@\space#2}%
1126     \else
1127       \bbl@vforeach{#1}{%
1128         \def\bbl@tempa{##1}%
1129         \bbl@fixname\bbl@tempa
1130         \bbl@iflanguage\bbl@tempa{%
1131           \bbl@csarg\protected@edef{hyphenation@\bbl@tempa}{%
1132             \bbl@ifunset{bbl@hyphenation@\bbl@tempa}%
1133               \@empty
1134               {\csname bbl@hyphenation@\bbl@tempa\endcsname\space}%
1135           #2}}}%
1136     \fi}}
```

\bbl@allowhyphens This macro makes hyphenation possible. Basically its definition is nothing more than
\nobreak \hskip 0pt plus 0pt[34].

---

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

```
1137 \def\bbl@allowhyphens{\ifvmode\else\nobreak\hskip\z@skip\fi}
1138 \def\bbl@t@one{T1}
1139 \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.

```
1140 \newcommand\babelnullhyphen{\char\hyphenchar\font}
1141 \def\babelhyphen{\active@prefix\babelhyphen\bbl@hyphen}
1142 \def\bbl@hyphen{%
1143   \@ifstar{\bbl@hyphen@i @}{\bbl@hyphen@i\@empty}}
1144 \def\bbl@hyphen@i#1#2{%
1145   \bbl@ifunset{bbl@hy@#1#2\@empty}%
1146     {\csname bbl@#1usehyphen\endcsname{\discretionary{#2}{}{#2}}}%
1147     {\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.

```
1148 \def\bbl@usehyphen#1{%
1149   \leavevmode
1150   \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
1151   \nobreak\hskip\z@skip}
1152 \def\bbl@@usehyphen#1{%
1153   \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}
```

The following macro inserts the hyphen char.

```
1154 \def\bbl@hyphenchar{%
1155   \ifnum\hyphenchar\font=\m@ne
1156     \babelnullhyphen
1157   \else
1158     \char\hyphenchar\font
1159   \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.

```
1160 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}{}}}
1161 \def\bbl@hy@@soft{\bbl@@usehyphen{\discretionary{\bbl@hyphenchar}{}{}}}
1162 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
1163 \def\bbl@hy@@hard{\bbl@@usehyphen\bbl@hyphenchar}
1164 \def\bbl@hy@nobreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}}
1165 \def\bbl@hy@@nobreak{\mbox{\bbl@hyphenchar}}
1166 \def\bbl@hy@repeat{%
1167   \bbl@usehyphen{%
1168     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
1169 \def\bbl@hy@@repeat{%
1170   \bbl@@usehyphen{%
1171     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
1172 \def\bbl@hy@empty{\hskip\z@skip}
1173 \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.

```
1174 \def\bbl@disc#1#2{\nobreak\discretionary{#2-}{}{#1}\bbl@allowhyphens}
```

9.9 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.

```
1175 \bbl@trace{Multiencoding strings}
1176 \def\bbl@toglobal#1{\global\let#1#1}
1177 \def\bbl@recatcode#1{%
1178   \@tempcnta="7F
1179   \def\bbl@tempa{%
1180     \ifnum\@tempcnta>"FF\else
1181       \catcode\@tempcnta=#1\relax
1182       \advance\@tempcnta\@ne
1183       \expandafter\bbl@tempa
1184     \fi}%
1185   \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).

```
1186 \@ifpackagewith{babel}{nocase}%
1187   {\let\bbl@patchuclc\relax}%
1188   {\def\bbl@patchuclc{%
1189     \global\let\bbl@patchuclc\relax
1190     \g@addto@macro\@uclclist{\reserved@b{\reserved@b\bbl@uclc}}%
1191     \gdef\bbl@uclc##1{%
1192       \let\bbl@encoded\bbl@encoded@uclc
1193       \bbl@ifunset{\languagename @bbl@uclc}% and resumes it
1194         {##1}%
1195         {\let\bbl@tempa##1\relax % Used by LANG@bbl@uclc
1196          \csname\languagename @bbl@uclc\endcsname}%
1197       {\bbl@tolower\@empty}{\bbl@toupper\@empty}}%
1198     \gdef\bbl@tolower{\csname\languagename @bbl@lc\endcsname}%
1199     \gdef\bbl@toupper{\csname\languagename @bbl@uc\endcsname}}}
```

```
1200 ⟨⟨*More package options⟩⟩ ≡
1201 \DeclareOption{nocase}{}
1202 ⟨⟨/More package options⟩⟩
```

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

```
1203 ⟨⟨*More package options⟩⟩ ≡
1204 \let\bbl@opt@strings\@nnil % accept strings=value
1205 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
1206 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
1207 \def\BabelStringsDefault{generic}
1208 ⟨⟨/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.

```
1209 \@onlypreamble\StartBabelCommands
1210 \def\StartBabelCommands{%
1211   \begingroup
1212   \bbl@recatcode{11}%
1213   ⟨⟨Macros local to BabelCommands⟩⟩
1214   \def\bbl@provstring##1##2{%
1215     \providecommand##1{##2}%
1216     \bbl@toglobal##1}%
1217   \global\let\bbl@scafter\@empty
1218   \let\StartBabelCommands\bbl@startcmds
1219   \ifx\BabelLanguages\relax
1220       \let\BabelLanguages\CurrentOption
1221   \fi
1222   \begingroup
1223   \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
1224   \StartBabelCommands}
1225 \def\bbl@startcmds{%
1226   \ifx\bbl@screset\@nnil\else
1227     \bbl@usehooks{stopcommands}{}%
1228   \fi
1229   \endgroup
1230   \begingroup
1231   \@ifstar
1232     {\ifx\bbl@opt@strings\@nnil
1233         \let\bbl@opt@strings\BabelStringsDefault
1234      \fi
1235      \bbl@startcmds@i}%
1236     \bbl@startcmds@i}
1237 \def\bbl@startcmds@i#1#2{%
1238   \edef\bbl@L{\zap@space#1 \@empty}%
1239   \edef\bbl@G{\zap@space#2 \@empty}%
1240   \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.

```
1241 \newcommand\bbl@startcmds@ii[1][\@empty]{%
1242   \let\SetString\@gobbletwo
1243   \let\bbl@stringdef\@gobbletwo
1244   \let\AfterBabelCommands\@gobble
1245   \ifx\@empty#1%
1246     \def\bbl@sc@label{generic}%
1247     \def\bbl@encstring##1##2{%
1248       \ProvideTextCommandDefault##1{##2}%
1249       \bbl@toglobal##1%
1250       \expandafter\bbl@toglobal\csname\string?\string##1\endcsname}%
1251     \let\bbl@sctest\in@true
1252   \else
```

```
1253     \let\bbl@sc@charset\space % <- zapped below
1254     \let\bbl@sc@fontenc\space % <-    "       "
1255     \def\bbl@tempa##1=##2\@nil{%
1256       \bbl@csarg\edef{sc@\zap@space##1 \@empty}{##2 }}%
1257     \bbl@vforeach{label=#1}{\bbl@tempa##1\@nil}%
1258     \def\bbl@tempa##1 ##2{% space -> comma
1259       ##1%
1260       \ifx\@empty##2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa##2\fi}%
1261     \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%
1262     \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%
1263     \edef\bbl@sc@charset{\expandafter\zap@space\bbl@sc@charset\@empty}%
1264     \def\bbl@encstring##1##2{%
1265       \bbl@foreach\bbl@sc@fontenc{%
1266         \bbl@ifunset{T@####1}%
1267           {}%
1268           {\ProvideTextCommand##1{####1}{##2}%
1269            \bbl@toglobal##1%
1270            \expandafter
1271            \bbl@toglobal\csname####1\string##1\endcsname}}}%
1272     \def\bbl@sctest{%
1273       \bbl@xin@{,\bbl@opt@strings,}{,\bbl@sc@label,\bbl@sc@fontenc,}}%
1274   \fi
1275   \ifx\bbl@opt@strings\@nnil      % ie, no strings key -> defaults
1276   \else\ifx\bbl@opt@strings\relax   % ie, strings=encoded
1277     \let\AfterBabelCommands\bbl@aftercmds
1278     \let\SetString\bbl@setstring
1279     \let\bbl@stringdef\bbl@encstring
1280   \else      % ie, strings=value
1281   \bbl@sctest
1282   \ifin@
1283     \let\AfterBabelCommands\bbl@aftercmds
1284     \let\SetString\bbl@setstring
1285     \let\bbl@stringdef\bbl@provstring
1286   \fi\fi\fi
1287   \bbl@scswitch
1288   \ifx\bbl@G\@empty
1289     \def\SetString##1##2{%
1290       \bbl@error{Missing group for string \string##1}%
1291         {You must assign strings to some category, typically\\%
1292          captions or extras, but you set none}}%
1293   \fi
1294   \ifx\@empty#1%
1295     \bbl@usehooks{defaultcommands}{}%
1296   \else
1297     \@expandtwoargs
1298     \bbl@usehooks{encodedcommands}{{\bbl@sc@charset}{\bbl@sc@fontenc}}%
1299   \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) .

```
1300 \def\bbl@forlang#1#2{%
1301   \bbl@for#1\bbl@L{%
```

```
1302     \bbl@xin@{,#1,}{,\BabelLanguages,}%
1303     \ifin@#2\relax\fi}}
1304 \def\bbl@scswitch{%
1305   \bbl@forlang\bbl@tempa{%
1306     \ifx\bbl@G\@empty\else
1307       \ifx\SetString\@gobbletwo\else
1308         \edef\bbl@GL{\bbl@G\bbl@tempa}%
1309         \bbl@xin@{,\bbl@GL,}{,\bbl@screset,}%
1310         \ifin@\else
1311           \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
1312           \xdef\bbl@screset{\bbl@screset,\bbl@GL}%
1313         \fi
1314       \fi
1315     \fi}}
1316 \AtEndOfPackage{%
1317   \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{}{#2}}}%
1318   \let\bbl@scswitch\relax}
1319 \@onlypreamble\EndBabelCommands
1320 \def\EndBabelCommands{%
1321   \bbl@usehooks{stopcommands}{}%
1322   \endgroup
1323   \endgroup
1324   \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.

```
1325 \def\bbl@setstring#1#2{%
1326   \bbl@forlang\bbl@tempa{%
1327     \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
1328     \bbl@ifunset{\bbl@LC}%  eg, \germanchaptername
1329       {\global\expandafter  % TODO - con \bbl@exp ?
1330        \bbl@add\csname\bbl@G\bbl@tempa\expandafter\endcsname\expandafter
1331          {\expandafter\bbl@scset\expandafter#1\csname\bbl@LC\endcsname}}%
1332       {}%
1333     \def\BabelString{#2}%
1334     \bbl@usehooks{stringprocess}{}%
1335     \expandafter\bbl@stringdef
1336       \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.

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

```
1347        \csname ?\string#1\endcsname
1348      \fi
1349    \else
1350      \csname\cf@encoding\string#1\endcsname
1351    \fi}
1352 \else
1353   \def\bbl@scset#1#2{\def#1{#2}}
1354 \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.

```
1355 ⟨⟨∗Macros local to BabelCommands⟩⟩ ≡
1356 \def\SetStringLoop##1##2{%
1357     \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
1358     \count@\z@
1359     \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
1360       \advance\count@\@ne
1361       \toks@\expandafter{\bbl@tempa}%
1362       \bbl@exp{%
1363         \\\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
1364         \count@=\the\count@\relax}}}%
1365 ⟨⟨/Macros local to BabelCommands⟩⟩
```

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

```
1366 \def\bbl@aftercmds#1{%
1367   \toks@\expandafter{\bbl@scafter#1}%
1368   \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.

```
1369 ⟨⟨∗Macros local to BabelCommands⟩⟩ ≡
1370   \newcommand\SetCase[3][]{%
1371     \bbl@patchuclc
1372     \bbl@forlang\bbl@tempa{%
1373       \expandafter\bbl@encstring
1374         \csname\bbl@tempa @bbl@uclc\endcsname{\bbl@tempa##1}%
1375       \expandafter\bbl@encstring
1376         \csname\bbl@tempa @bbl@uc\endcsname{##2}%
1377       \expandafter\bbl@encstring
1378         \csname\bbl@tempa @bbl@lc\endcsname{##3}}}%
1379 ⟨⟨/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.

```
1380 ⟨⟨∗Macros local to BabelCommands⟩⟩ ≡
1381   \newcommand\SetHyphenMap[1]{%
1382     \bbl@forlang\bbl@tempa{%
1383       \expandafter\bbl@stringdef
1384         \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}
1385 ⟨⟨/Macros local to BabelCommands⟩⟩
```

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

```
1386 \newcommand\BabelLower[2]{% one to one.
1387   \ifnum\lccode#1=#2\else
1388     \babel@savevariable{\lccode#1}%
1389     \lccode#1=#2\relax
1390   \fi}
1391 \newcommand\BabelLowerMM[4]{% many-to-many
1392   \@tempcnta=#1\relax
1393   \@tempcntb=#4\relax
1394   \def\bbl@tempa{%
1395     \ifnum\@tempcnta>#2\else
1396       \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
1397       \advance\@tempcnta#3\relax
1398       \advance\@tempcntb#3\relax
1399       \expandafter\bbl@tempa
1400     \fi}%
1401   \bbl@tempa}
1402 \newcommand\BabelLowerMO[4]{% many-to-one
1403   \@tempcnta=#1\relax
1404   \def\bbl@tempa{%
1405     \ifnum\@tempcnta>#2\else
1406       \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
1407       \advance\@tempcnta#3
1408       \expandafter\bbl@tempa
1409     \fi}%
1410   \bbl@tempa}
```

The following package options control the behavior of hyphenation mapping.

```
1411 ⟨⟨∗More package options⟩⟩ ≡
1412 \DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\z@}
1413 \DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap\@ne}
1414 \DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\tw@}
1415 \DeclareOption{hyphenmap=other}{\chardef\bbl@opt@hyphenmap\thr@@}
1416 \DeclareOption{hyphenmap=other*}{\chardef\bbl@opt@hyphenmap4\relax}
1417 ⟨⟨/More package options⟩⟩
```

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

```
1418 \AtEndOfPackage{%
1419   \ifx\bbl@opt@hyphenmap\@undefined
1420     \bbl@xin@{,}{\bbl@language@opts}%
1421     \chardef\bbl@opt@hyphenmap\ifin@4\else\@ne\fi
1422   \fi}
```

9.10   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.

```
1423 \bbl@trace{Macros related to glyphs}
1424 \def\set@low@box#1{\setbox\tw@\hbox{,}\setbox\z@\hbox{#1}%
1425   \dimen\z@\ht\z@ \advance\dimen\z@ -\ht\tw@%
1426   \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.

```
1427 \def\save@sf@q#1{\leavevmode
1428   \begingroup
1429     \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
1430   \endgroup}
```

9.11 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.

9.11.1 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.

```
1431 \ProvideTextCommand{\quotedblbase}{OT1}{%
1432   \save@sf@q{\set@low@box{\textquotedblright\/}%
1433     \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.

```
1434 \ProvideTextCommandDefault{\quotedblbase}{%
1435   \UseTextSymbol{OT1}{\quotedblbase}}
```

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

```
1436 \ProvideTextCommand{\quotesinglbase}{OT1}{%
1437   \save@sf@q{\set@low@box{\textquoteright\/}%
1438     \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.

```
1439 \ProvideTextCommandDefault{\quotesinglbase}{%
1440   \UseTextSymbol{OT1}{\quotesinglbase}}
```

\guillemotleft   The guillemet characters are not available in OT1 encoding. They are faked.
\guillemotright
```
1441 \ProvideTextCommand{\guillemotleft}{OT1}{%
1442   \ifmmode
1443     \ll
1444   \else
1445     \save@sf@q{\nobreak
1446       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
1447   \fi}
1448 \ProvideTextCommand{\guillemotright}{OT1}{%
1449   \ifmmode
1450     \gg
1451   \else
1452     \save@sf@q{\nobreak
1453       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
1454   \fi}
```

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

```
1455 \ProvideTextCommandDefault{\guillemotleft}{%
1456   \UseTextSymbol{OT1}{\guillemotleft}}
1457 \ProvideTextCommandDefault{\guillemotright}{%
1458   \UseTextSymbol{OT1}{\guillemotright}}
```

\guilsinglleft   The single guillemets are not available in OT1 encoding. They are faked.
\guilsinglright
```
1459 \ProvideTextCommand{\guilsinglleft}{OT1}{%
1460   \ifmmode
1461     <%
1462   \else
```

90

```
1463    \save@sf@q{\nobreak
1464      \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
1465    \fi}
1466 \ProvideTextCommand{\guilsinglright}{OT1}{%
1467    \ifmmode
1468      >%
1469    \else
1470      \save@sf@q{\nobreak
1471        \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
1472    \fi}
```

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

```
1473 \ProvideTextCommandDefault{\guilsinglleft}{%
1474    \UseTextSymbol{OT1}{\guilsinglleft}}
1475 \ProvideTextCommandDefault{\guilsinglright}{%
1476    \UseTextSymbol{OT1}{\guilsinglright}}
```

9.11.2 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.

```
1477 \DeclareTextCommand{\ij}{OT1}{%
1478    i\kern-0.02em\bbl@allowhyphens j}
1479 \DeclareTextCommand{\IJ}{OT1}{%
1480    I\kern-0.02em\bbl@allowhyphens J}
1481 \DeclareTextCommand{\ij}{T1}{\char188}
1482 \DeclareTextCommand{\IJ}{T1}{\char156}
```

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

```
1483 \ProvideTextCommandDefault{\ij}{%
1484    \UseTextSymbol{OT1}{\ij}}
1485 \ProvideTextCommandDefault{\IJ}{%
1486    \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).

```
1487 \def\crrtic@{\hrule height0.1ex width0.3em}
1488 \def\crttic@{\hrule height0.1ex width0.33em}
1489 \def\ddj@{%
1490    \setbox0\hbox{d}\dimen@=\ht0
1491    \advance\dimen@1ex
1492    \dimen@.45\dimen@
1493    \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
1494    \advance\dimen@ii.5ex
1495    \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
1496 \def\DDJ@{%
1497    \setbox0\hbox{D}\dimen@=.55\ht0
1498    \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
1499    \advance\dimen@ii.15ex %              correction for the dash position
1500    \advance\dimen@ii-.15\fontdimen7\font %    correction for cmtt font
1501    \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
1502    \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
1503 %
```

```
1504 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
1505 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}
```

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

```
1506 \ProvideTextCommandDefault{\dj}{%
1507     \UseTextSymbol{OT1}{\dj}}
1508 \ProvideTextCommandDefault{\DJ}{%
1509     \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.

```
1510 \DeclareTextCommand{\SS}{OT1}{SS}
1511 \ProvideTextCommandDefault{\SS}{\UseTextSymbol{OT1}{\SS}}
```

9.11.3   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
```
1512 \ProvideTextCommandDefault{\glq}{%
1513     \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}
```

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

```
1514 \ProvideTextCommand{\grq}{T1}{%
1515     \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
1516 \ProvideTextCommand{\grq}{TU}{%
1517     \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
1518 \ProvideTextCommand{\grq}{OT1}{%
1519     \save@sf@q{\kern-.0125em
1520       \textormath{\textquoteleft}{\mbox{\textquoteleft}}%
1521       \kern.07em\relax}}
1522 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}\grq}
```

\glqq   The 'german' double quotes.

\grqq
```
1523 \ProvideTextCommandDefault{\glqq}{%
1524     \textormath{\quotedblbase}{\mbox{\quotedblbase}}}
```

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

```
1525 \ProvideTextCommand{\grqq}{T1}{%
1526     \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
1527 \ProvideTextCommand{\grqq}{TU}{%
1528     \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
1529 \ProvideTextCommand{\grqq}{OT1}{%
1530     \save@sf@q{\kern-.07em
1531       \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}%
1532       \kern.07em\relax}}
1533 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}
```

\flq   The 'french' single guillemets.

\frq
```
1534 \ProvideTextCommandDefault{\flq}{%
1535     \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
1536 \ProvideTextCommandDefault{\frq}{%
1537     \textormath{\guilsinglright}{\mbox{\guilsinglright}}}
```

\flqq   The 'french' double guillemets.

\frqq
```
1538 \ProvideTextCommandDefault{\flqq}{%
1539   \textormath{\guillemotleft}{\mbox{\guillemotleft}}}
1540 \ProvideTextCommandDefault{\frqq}{%
1541   \textormath{\guillemotright}{\mbox{\guillemotright}}}
```

9.11.4 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).

```
1542 \def\umlauthigh{%
1543   \def\bbl@umlauta##1{\leavevmode\bgroup%
1544     \expandafter\accent\csname\f@encoding dqpos\endcsname
1545     ##1\bbl@allowhyphens\egroup}%
1546   \let\bbl@umlaute\bbl@umlauta}
1547 \def\umlautlow{%
1548   \def\bbl@umlauta{\protect\lower@umlaut}}
1549 \def\umlautelow{%
1550   \def\bbl@umlaute{\protect\lower@umlaut}}
1551 \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.

```
1552 \expandafter\ifx\csname U@D\endcsname\relax
1553   \csname newdimen\endcsname\U@D
1554 \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.

```
1555 \def\lower@umlaut#1{%
1556   \leavevmode\bgroup
1557     \U@D 1ex%
1558     {\setbox\z@\hbox{%
1559       \expandafter\char\csname\f@encoding dqpos\endcsname}%
1560       \dimen@ -.45ex\advance\dimen@\ht\z@
1561       \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}%
1562     \expandafter\accent\csname\f@encoding dqpos\endcsname
1563     \fontdimen5\font\U@D #1%
1564   \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).

```
1565 \AtBeginDocument{%
1566   \DeclareTextCompositeCommand{\"}{OT1}{a}{\bbl@umlauta{a}}%
1567   \DeclareTextCompositeCommand{\"}{OT1}{e}{\bbl@umlaute{e}}%
1568   \DeclareTextCompositeCommand{\"}{OT1}{i}{\bbl@umlaute{\i}}%
1569   \DeclareTextCompositeCommand{\"}{OT1}{\i}{\bbl@umlaute{\i}}%
1570   \DeclareTextCompositeCommand{\"}{OT1}{o}{\bbl@umlauta{o}}%
1571   \DeclareTextCompositeCommand{\"}{OT1}{u}{\bbl@umlauta{u}}%
1572   \DeclareTextCompositeCommand{\"}{OT1}{A}{\bbl@umlauta{A}}%
1573   \DeclareTextCompositeCommand{\"}{OT1}{E}{\bbl@umlaute{E}}%
1574   \DeclareTextCompositeCommand{\"}{OT1}{I}{\bbl@umlaute{I}}%
1575   \DeclareTextCompositeCommand{\"}{OT1}{O}{\bbl@umlauta{O}}%
1576   \DeclareTextCompositeCommand{\"}{OT1}{U}{\bbl@umlauta{U}}%
1577 }
```

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

```
1578 \ifx\l@english\@undefined
1579   \chardef\l@english\z@
1580 \fi
1581 \main@language{english}
```

9.12 Layout

Work in progress.

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

```
1582 \bbl@trace{Bidi layout}
1583 \providecommand\IfBabelLayout[3]{#3}%
1584 \newcommand\BabelPatchSection[1]{%
1585   \@ifundefined{#1}{}{%
1586     \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
1587     \@namedef{#1}{%
1588       \@ifstar{\bbl@presec@s{#1}}%
1589               {\@dblarg{\bbl@presec@x{#1}}}}}}
1590 \def\bbl@presec@x#1[#2]#3{%
1591   \bbl@exp{%
1592     \\\select@language@x{\bbl@main@language}%
1593     \\\@nameuse{bbl@sspre@#1}%
1594     \\\@nameuse{bbl@ss@#1}%
1595       [\\\foreignlanguage{\languagename}{\unexpanded{#2}}]%
1596       {\\\foreignlanguage{\languagename}{\unexpanded{#3}}}%
1597     \\\select@language@x{\languagename}}}
1598 \def\bbl@presec@s#1#2{%
1599   \bbl@exp{%
1600     \\\select@language@x{\bbl@main@language}%
1601     \\\@nameuse{bbl@sspre@#1}%
1602     \\\@nameuse{bbl@ss@#1}*%
1603       {\\\foreignlanguage{\languagename}{\unexpanded{#2}}}%
1604     \\\select@language@x{\languagename}}}
1605 \IfBabelLayout{sectioning}%
1606   {\BabelPatchSection{part}%
1607     \BabelPatchSection{chapter}%
1608     \BabelPatchSection{section}%
1609     \BabelPatchSection{subsection}%
```

```
1610    \BabelPatchSection{subsubsection}%
1611    \BabelPatchSection{paragraph}%
1612    \BabelPatchSection{subparagraph}%
1613    \def\babel@toc#1{%
1614      \select@language@x{\bbl@main@language}}}{}
1615 \IfBabelLayout{captions}%
1616   {\BabelPatchSection{caption}}{}
```

Now we load definition files for engines.

```
1617 \bbl@trace{Input engine specific macros}
1618 \ifcase\bbl@engine
1619    \input txtbabel.def
1620 \or
1621    \input luababel.def
1622 \or
1623    \input xebabel.def
1624 \fi
```

9.13   Creating languages

`\babelprovide` is a general purpose tool for creating and modifying languages. It creates the language infrastructure, and loads, if requested, an ini file. It may be used in conjunction to previouly loaded `ldf` files.

```
1625 \bbl@trace{Creating languages and reading ini files}
1626 \newcommand\babelprovide[2][]{%
1627    \let\bbl@savelangname\languagename
1628    \def\languagename{#2}%
1629    \let\bbl@KVP@captions\@nil
1630    \let\bbl@KVP@import\@nil
1631    \let\bbl@KVP@main\@nil
1632    \let\bbl@KVP@script\@nil
1633    \let\bbl@KVP@language\@nil
1634    \let\bbl@KVP@dir\@nil
1635    \let\bbl@KVP@hyphenrules\@nil
1636    \let\bbl@KVP@mapfont\@nil
1637    \let\bbl@KVP@maparabic\@nil
1638    \let\bbl@KVP@intraspace\@nil
1639    \let\bbl@KVP@intrapenalty\@nil
1640    \bbl@forkv{#1}{\bbl@csarg\def{KVP@##1}{##2}}%  TODO - error handling
1641    \ifx\bbl@KVP@import\@nil\else
1642      \bbl@exp{\\\bbl@ifblank{\bbl@KVP@import}}%
1643        {\begingroup
1644          \def\BabelBeforeIni##1##2{\gdef\bbl@KVP@import{##1}\endinput}%
1645          \InputIfFileExists{babel-#2.tex}{}{}%
1646         \endgroup}%
1647        {}%
1648    \fi
1649    \ifx\bbl@KVP@captions\@nil
1650      \let\bbl@KVP@captions\bbl@KVP@import
1651    \fi
1652 % Load ini
1653    \bbl@ifunset{date#2}%
1654      {\bbl@provide@new{#2}}%
1655      {\bbl@ifblank{#1}%
1656        {\bbl@error
1657          {If you want to modify `#2' you must tell how in\\%
1658           the optional argument. See the manual for the\\%
1659           available options.}%
```

```
1660          {Use this macro as documented}}%
1661        {\bbl@provide@renew{#2}}}%
1662    % Post tasks
1663    \bbl@exp{\\\babelensure[exclude=\\\today]{#2}}%
1664    \bbl@ifunset{bbl@ensure@\languagename}%
1665      {\bbl@exp{%
1666        \\\DeclareRobustCommand\<bbl@ensure@\languagename>[1]{%
1667          \\\foreignlanguage{\languagename}%
1668          {####1}}}}%
1669      {}%
1670    % To override script and language names
1671    \ifx\bbl@KVP@script\@nil\else
1672      \bbl@csarg\edef{sname@#2}{\bbl@KVP@script}%
1673    \fi
1674    \ifx\bbl@KVP@language\@nil\else
1675      \bbl@csarg\edef{lname@#2}{\bbl@KVP@language}%
1676    \fi
1677    % For bidi texts, to switch the language based on direction
1678    \ifx\bbl@KVP@mapfont\@nil\else
1679      \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}{}%
1680        {\bbl@error{Option `\bbl@KVP@mapfont' unknown for\\%
1681                     mapfont. Use `direction'.%
1682                    {See the manual for details.}}}%
1683      \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
1684      \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
1685      \ifx\bbl@mapselect\@undefined
1686        \AtBeginDocument{%
1687          \expandafter\bbl@add\csname selectfont \endcsname{{\bbl@mapselect}}%
1688          {\selectfont}}%
1689        \def\bbl@mapselect{%
1690          \let\bbl@mapselect\relax
1691          \edef\bbl@prefontid{\fontid\font}}%
1692        \def\bbl@mapdir##1{%
1693          {\def\languagename{##1}%
1694           \let\bbl@ifrestoring\@firstoftwo % avoid font warning
1695           \bbl@switchfont
1696           \directlua{Babel.fontmap
1697             [\the\csname bbl@wdir@##1\endcsname]%
1698             [\bbl@prefontid]=\fontid\font}}}%
1699      \fi
1700      \bbl@exp{\\\bbl@add\\\bbl@mapselect{\\\bbl@mapdir{\languagename}}}%
1701    \fi
1702    % For Southeast Asian, if interspace in ini
1703    \ifcase\bbl@engine\or
1704      \bbl@ifunset{bbl@intsp@\languagename}{}%
1705        {\expandafter\ifx\csname bbl@intsp@\languagename\endcsname\@empty\else
1706          \bbl@seaintraspace
1707          \ifx\bbl@KVP@intraspace\@nil
1708            \bbl@exp{%
1709              \\\bbl@intraspace\bbl@cs{intsp@\languagename}\\\@@}%
1710          \fi
1711          \directlua{
1712            Babel = Babel or {}
1713            Babel.sea_ranges = Babel.sea_ranges or {}
1714            Babel.set_chranges('\bbl@cs{sbcp@\languagename}',
1715                              '\bbl@cs{chrng@\languagename}')
1716          }
1717          \ifx\bbl@KVP@intrapenalty\@nil
1718            \bbl@intrapenalty0\@@
```

96

```
1719          \fi
1720        \fi
1721        \ifx\bbl@KVP@intraspace\@nil\else % We may override the ini
1722          \expandafter\bbl@intraspace\bbl@KVP@intraspace\@@
1723        \fi
1724        \ifx\bbl@KVP@intrapenalty\@nil\else
1725          \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
1726        \fi}%
1727   \or
1728      \bbl@xin@{\bbl@cs{sbcp@\languagename}}{Thai,Laoo,Khmr}%
1729      \ifin@
1730        \bbl@ifunset{bbl@intsp@\languagename}{}%
1731          {\expandafter\ifx\csname bbl@intsp@\languagename\endcsname\@empty\else
1732            \ifx\bbl@KVP@intraspace\@nil
1733              \bbl@exp{%
1734                \\\bbl@intraspace\bbl@cs{intsp@\languagename}\\\@@}%
1735            \fi
1736            \ifx\bbl@KVP@intrapenalty\@nil
1737              \bbl@intrapenalty0\@@
1738            \fi
1739          \fi
1740          \ifx\bbl@KVP@intraspace\@nil\else % We may override the ini
1741            \expandafter\bbl@intraspace\bbl@KVP@intraspace\@@
1742          \fi
1743          \ifx\bbl@KVP@intrapenalty\@nil\else
1744            \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
1745          \fi
1746          \ifx\bbl@ispacesize\@undefined
1747            \AtBeginDocument{%
1748              \expandafter\bbl@add
1749              \csname selectfont \endcsname{\bbl@ispacesize}}%
1750            \def\bbl@ispacesize{\bbl@cs{xeisp@\bbl@cs{sbcp@\languagename}}}%
1751        \fi}%
1752      \fi
1753   \fi
1754   % Native digits, if provided in ini
1755   \ifcase\bbl@engine\else
1756     \bbl@ifunset{bbl@dgnat@\languagename}{}%
1757       {\expandafter\ifx\csname bbl@dgnat@\languagename\endcsname\@empty\else
1758         \expandafter\expandafter\expandafter
1759         \bbl@setdigits\csname bbl@dgnat@\languagename\endcsname
1760         \ifx\bbl@KVP@maparabic\@nil\else
1761           \ifx\bbl@latinarabic\@undefined
1762             \expandafter\let\expandafter\@arabic
1763               \csname bbl@counter@\languagename\endcsname
1764           \else    % ie, if layout=counters, which redefines \@arabic
1765             \expandafter\let\expandafter\bbl@latinarabic
1766               \csname bbl@counter@\languagename\endcsname
1767           \fi
1768         \fi
1769       \fi}%
1770   \fi
1771   % To load or reaload the babel-*.tex, if require.babel in ini
1772   \bbl@ifunset{bbl@rqtex@\languagename}{}%
1773     {\expandafter\ifx\csname bbl@rqtex@\languagename\endcsname\@empty\else
1774       \let\BabelBeforeIni\@gobbletwo
1775       \chardef\atcatcode=\catcode`\@
1776       \catcode`\@=11\relax
1777       \InputIfFileExists{babel-\bbl@cs{rqtex@\languagename}.tex}{}{}%
```

```
1778        \catcode`\@=\atcatcode
1779        \let\atcatcode\relax
1780      \fi}%
1781   \let\languagename\bbl@savelangname}
```

A tool to define the macros for native digits from the list provided in the ini file. Somewhat convoluted because there are 10 digits, but only 9 arguments in TEX.

```
1782 \def\bbl@setdigits#1#2#3#4#5{%
1783   \bbl@exp{%
1784     \def\<\languagename digits>####1{%          ie, \langdigits
1785       \<bbl@digits@\languagename>####1\\\@nil}%
1786     \def\<\languagename counter>####1{%          ie, \langcounter
1787       \\\expandafter\<bbl@counter@\languagename>%
1788       \\\csname c@####1\endcsname}%
1789     \def\<bbl@counter@\languagename>####1{% ie, \bbl@counter@lang
1790       \\\expandafter\<bbl@digits@\languagename>%
1791       \\\number####1\\\@nil}}%
1792   \def\bbl@tempa##1##2##3##4##5{%
1793     \bbl@exp{%     Wow, quite a lot of hashes! :-(
1794       \def\<bbl@digits@\languagename>########1{%
1795       \\\ifx########1\\\@nil               % ie, \bbl@digits@lang
1796       \\\else
1797         \\\ifx0########1#1%
1798         \\\else\\\ifx1########1#2%
1799         \\\else\\\ifx2########1#3%
1800         \\\else\\\ifx3########1#4%
1801         \\\else\\\ifx4########1#5%
1802         \\\else\\\ifx5########1##1%
1803         \\\else\\\ifx6########1##2%
1804         \\\else\\\ifx7########1##3%
1805         \\\else\\\ifx8########1##4%
1806         \\\else\\\ifx9########1##5%
1807         \\\else########1%
1808         \\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi
1809         \\\expandafter\<bbl@digits@\languagename>%
1810       \\\fi}}}%
1811   \bbl@tempa}
```

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

```
1812 \def\bbl@provide@new#1{%
1813   \@namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
1814   \@namedef{extras#1}{}%
1815   \@namedef{noextras#1}{}%
1816   \StartBabelCommands*{#1}{captions}%
1817     \ifx\bbl@KVP@captions\@nil %      and also if import, implicit
1818       \def\bbl@tempb##1{%               elt for \bbl@captionslist
1819         \ifx##1\@empty\else
1820           \bbl@exp{%
1821             \\\SetString\\##1{%
1822               \\\bbl@nocaption{\bbl@stripslash##1}{#1\bbl@stripslash##1}}}%
1823           \expandafter\bbl@tempb
1824         \fi}%
1825       \expandafter\bbl@tempb\bbl@captionslist\@empty
1826     \else
1827       \bbl@read@ini{\bbl@KVP@captions}%  Here all letters cat = 11
1828       \bbl@after@ini
1829       \bbl@savestrings
1830     \fi
```

98

```
1831   \StartBabelCommands*{#1}{date}%
1832     \ifx\bbl@KVP@import\@nil
1833       \bbl@exp{%
1834         \\\SetString\\\today{\\\bbl@nocaption{today}{#1today}}}%
1835     \else
1836       \bbl@savetoday
1837       \bbl@savedate
1838     \fi
1839   \EndBabelCommands
1840   \bbl@exp{%
1841     \def\<#1hyphenmins>{%
1842       {\bbl@ifunset{bbl@lfthm@#1}{2}{\@nameuse{bbl@lfthm@#1}}}%
1843       {\bbl@ifunset{bbl@rgthm@#1}{3}{\@nameuse{bbl@rgthm@#1}}}}}%
1844   \bbl@provide@hyphens{#1}%
1845   \ifx\bbl@KVP@main\@nil\else
1846     \expandafter\main@language\expandafter{#1}%
1847   \fi}
1848 \def\bbl@provide@renew#1{%
1849   \ifx\bbl@KVP@captions\@nil\else
1850     \StartBabelCommands*{#1}{captions}%
1851       \bbl@read@ini{\bbl@KVP@captions}%   Here all letters cat = 11
1852       \bbl@after@ini
1853       \bbl@savestrings
1854     \EndBabelCommands
1855   \fi
1856   \ifx\bbl@KVP@import\@nil\else
1857     \StartBabelCommands*{#1}{date}%
1858       \bbl@savetoday
1859       \bbl@savedate
1860     \EndBabelCommands
1861   \fi
1862   \bbl@provide@hyphens{#1}}
```

The hyphenrules option is handled with an auxiliary macro.

```
1863 \def\bbl@provide@hyphens#1{%
1864   \let\bbl@tempa\relax
1865   \ifx\bbl@KVP@hyphenrules\@nil\else
1866     \bbl@replace\bbl@KVP@hyphenrules{ }{,}%
1867     \bbl@foreach\bbl@KVP@hyphenrules{%
1868       \ifx\bbl@tempa\relax    % if not yet found
1869         \bbl@ifsamestring{##1}{+}%
1870           {{\bbl@exp{\\\addlanguage\<l@##1>}}}%
1871           {}%
1872         \bbl@ifunset{l@##1}%
1873           {}%
1874           {\bbl@exp{\let\bbl@tempa\<l@##1>}}%
1875       \fi}%
1876   \fi
1877   \ifx\bbl@tempa\relax %        if no opt or no language in opt found
1878     \ifx\bbl@KVP@import\@nil\else % if importing
1879       \bbl@exp{%                and hyphenrules is not empty
1880         \\\bbl@ifblank{\@nameuse{bbl@hyphr@#1}}%
1881           {}%
1882           {\let\\\bbl@tempa\<l@\@nameuse{bbl@hyphr@\languagename}>}}%
1883     \fi
1884   \fi
1885   \bbl@ifunset{bbl@tempa}%       ie, relax or undefined
1886     {\bbl@ifunset{l@#1}%         no hyphenrules found - fallback
1887       {\bbl@exp{\\\adddialect\<l@#1>\language}}%
```

```
1888        {}}%                      so, l@<lang> is ok - nothing to do
1889     {\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.*

```
1890 \def\bbl@read@ini#1{%
1891   \openin1=babel-#1.ini
1892   \ifeof1
1893     \bbl@error
1894       {There is no ini file for the requested language\\%
1895        (#1). Perhaps you misspelled it or your installation\\%
1896        is not complete.}%
1897       {Fix the name or reinstall babel.}%
1898   \else
1899     \let\bbl@section\@empty
1900     \let\bbl@savestrings\@empty
1901     \let\bbl@savetoday\@empty
1902     \let\bbl@savedate\@empty
1903     \let\bbl@inireader\bbl@iniskip
1904     \bbl@info{Importing data from babel-#1.ini for \languagename}%
1905     \loop
1906     \if T\ifeof1F\fi T\relax % Trick, because inside \loop
1907       \endlinechar\m@ne
1908       \read1 to \bbl@line
1909       \endlinechar`\^^M
1910       \ifx\bbl@line\@empty\else
1911         \expandafter\bbl@iniline\bbl@line\bbl@iniline
1912       \fi
1913     \repeat
1914   \fi}
1915 \def\bbl@iniline#1\bbl@iniline{%
1916   \@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.

```
1917 \def\bbl@iniskip#1\@@{}%       if starts with ;
1918 \def\bbl@inisec[#1]#2\@@{%     if starts with opening bracket
1919   \@nameuse{bbl@secpost@\bbl@section}%  ends previous section
1920   \def\bbl@section{#1}%
1921   \@nameuse{bbl@secpre@\bbl@section}%   starts current section
1922   \bbl@ifunset{bbl@inikv@#1}%
1923     {\let\bbl@inireader\bbl@iniskip}%
1924     {\bbl@exp{\let\\\bbl@inireader\<bbl@inikv@#1>}}}
```

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

```
1925 \def\bbl@inikv#1=#2\@@{%       key=value
1926   \bbl@trim@def\bbl@tempa{#1}%
1927   \bbl@trim\toks@{#2}%
1928   \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.

```
1929 \def\bbl@exportkey#1#2#3{%
1930   \bbl@ifunset{bbl@@kv@#2}%
1931     {\bbl@csarg\gdef{#1@\languagename}{#3}}%
1932     {\expandafter\ifx\csname bbl@@kv@#2\endcsname\@empty
1933        \bbl@csarg\gdef{#1@\languagename}{#3}%
1934      \else
```

```
1935        \bbl@exp{\global\let\<bbl@#1@\languagename>\<bbl@@kv@#2>}%
1936      \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.

```
1937 \let\bbl@inikv@identification\bbl@inikv
1938 \def\bbl@secpost@identification{%
1939   \bbl@exportkey{lname}{identification.name.english}{}%
1940   \bbl@exportkey{lbcp}{identification.tag.bcp47}{}%
1941   \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
1942   \bbl@exportkey{sname}{identification.script.name}{}%
1943   \bbl@exportkey{sbcp}{identification.script.tag.bcp47}{}%
1944   \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}}
1945 \let\bbl@inikv@typography\bbl@inikv
1946 \let\bbl@inikv@characters\bbl@inikv
1947 \let\bbl@inikv@numbers\bbl@inikv
1948 \def\bbl@after@ini{%
1949   \bbl@exportkey{lfthm}{typography.lefthyphenmin}{2}%
1950   \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
1951   \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
1952   \bbl@exportkey{intsp}{typography.intraspace}{}%
1953   \bbl@exportkey{jstfy}{typography.justify}{w}%
1954   \bbl@exportkey{chrng}{characters.ranges}{}%
1955   \bbl@exportkey{dgnat}{numbers.digits.native}{}%
1956   \bbl@exportkey{rqtex}{identification.require.babel}{}%
1957   \bbl@xin@{0.5}{\@nameuse{bbl@@kv@identification.version}}%
1958   \ifin@
1959     \bbl@warning{%
1960       There are neither captions nor date in `\languagename'.\\%
1961       It may not be suitable for proper typesetting, and it\\%
1962       could change. Reported}%
1963   \fi
1964   \bbl@xin@{0.9}{\@nameuse{bbl@@kv@identification.version}}%
1965   \ifin@
1966     \bbl@warning{%
1967       The `\languagename' date format may not be suitable\\%
1968       for proper typesetting, and therefore it very likely will\\%
1969       change in a future release. Reported}%
1970   \fi
1971   \bbl@toglobal\bbl@savetoday
1972   \bbl@toglobal\bbl@savedate}
```

Now captions and captions.licr, depending on the engine. And below also for dates. They rely on a few auxiliary macros. It is expected the ini file provides the complete set in Unicode and LICR, in that order.

```
1973 \ifcase\bbl@engine
1974   \bbl@csarg\def{inikv@captions.licr}#1=#2\@@{%
1975     \bbl@ini@captions@aux{#1}{#2}}
1976 \else
1977   \def\bbl@inikv@captions#1=#2\@@{%
1978     \bbl@ini@captions@aux{#1}{#2}}
1979 \fi
```

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

```
1980 \def\bbl@ini@captions@aux#1#2{%
1981   \bbl@trim@def\bbl@tempa{#1}%
1982   \bbl@ifblank{#2}%
1983     {\bbl@exp{%
1984        \toks@{\\\bbl@nocaption{\bbl@tempa}{\languagename\bbl@tempa name}}}}%
```

101

```
1985        {\bbl@trim\toks@{#2}}%
1986    \bbl@exp{%
1987        \\\bbl@add\\\bbl@savestrings{%
1988            \\\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 (saved months are just discarded when the LICR section is reached).

TODO. Remove copypaste pattern.

```
1989 \bbl@csarg\def{inikv@date.gregorian}#1=#2\@@{%          for defaults
1990    \bbl@inidate#1...\relax{#2}{}}
1991 \bbl@csarg\def{inikv@date.islamic}#1=#2\@@{%
1992    \bbl@inidate#1...\relax{#2}{islamic}}
1993 \bbl@csarg\def{inikv@date.hebrew}#1=#2\@@{%
1994    \bbl@inidate#1...\relax{#2}{hebrew}}
1995 \bbl@csarg\def{inikv@date.persian}#1=#2\@@{%
1996    \bbl@inidate#1...\relax{#2}{persian}}
1997 \bbl@csarg\def{inikv@date.indian}#1=#2\@@{%
1998    \bbl@inidate#1...\relax{#2}{indian}}
1999 \ifcase\bbl@engine
2000    \bbl@csarg\def{inikv@date.gregorian.licr}#1=#2\@@{%  override
2001        \bbl@inidate#1...\relax{#2}{}}
2002    \bbl@csarg\def{secpre@date.gregorian.licr}{%             discard uni
2003        \ifcase\bbl@engine\let\bbl@savedate\@empty\fi}
2004 \fi
2005 % eg: 1=months, 2=wide, 3=1, 4=dummy
2006 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
2007    \bbl@trim@def\bbl@tempa{#1.#2}%
2008    \bbl@ifsamestring{\bbl@tempa}{months.wide}%       to savedate
2009        {\bbl@trim@def\bbl@tempa{#3}%
2010        \bbl@trim\toks@{#5}%
2011        \bbl@exp{%
2012            \\\bbl@add\\\bbl@savedate{%
2013                \\\SetString\<month\romannumeral\bbl@tempa#6name>{\the\toks@}}}}%
2014        {\bbl@ifsamestring{\bbl@tempa}{date.long}%       defined now
2015            {\bbl@trim@def\bbl@toreplace{#5}%
2016            \bbl@TG@@date
2017            \global\bbl@csarg\let{date@\languagename}\bbl@toreplace
2018            \bbl@exp{%
2019                \gdef\<\languagename date>{\\\protect\<\languagename date >}%
2020                \gdef\<\languagename date >####1####2####3{%
2021                    \\\bbl@usedategrouptrue
2022                    \<bbl@ensure@\languagename>{%
2023                        \<bbl@date@\languagename>{####1}{####2}{####3}}}%
2024                \\\bbl@add\\\bbl@savetoday{%
2025                    \\\SetString\\\today{%
2026                        \<\languagename date>{\\\the\year}{\\\the\month}{\\\the\day}}}}}}%
2027            {}}
```

Dates will require some macros for the basic formatting. They may be redefined by language, so "semi-public" names (camel case) are used. Oddly enough, the CLDR places particles like "de" inconsistently in either in the date or in the month name.

```
2028 \let\bbl@calendar\@empty
2029 \newcommand\BabelDateSpace{\nobreakspace}
2030 \newcommand\BabelDateDot{.\@}
2031 \newcommand\BabelDated[1]{{\number#1}}
2032 \newcommand\BabelDatedd[1]{{\ifnum#1<10 0\fi\number#1}}
2033 \newcommand\BabelDateM[1]{{\number#1}}
2034 \newcommand\BabelDateMM[1]{{\ifnum#1<10 0\fi\number#1}}
```

```
2035 \newcommand\BabelDateMMMM[1]{{%
2036   \csname month\romannumeral#1\bbl@calendar name\endcsname}}%
2037 \newcommand\BabelDatey[1]{{\number#1}}%
2038 \newcommand\BabelDateyy[1]{{%
2039   \ifnum#1<10 0\number#1 %
2040   \else\ifnum#1<100 \number#1 %
2041   \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
2042   \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
2043   \else
2044     \bbl@error
2045       {Currently two-digit years are restricted to the\\
2046        range 0-9999.}%
2047       {There is little you can do. Sorry.}%
2048   \fi\fi\fi\fi}}
2049 \newcommand\BabelDateyyyy[1]{{\number#1}} % FIXME - add leading 0
2050 \def\bbl@replace@finish@iii#1{%
2051   \bbl@exp{\def\\#1####1####2####3{\the\toks@}}}
2052 \def\bbl@TG@@date{%
2053   \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace{}}%
2054   \bbl@replace\bbl@toreplace{[.]}{\BabelDateDot{}}%
2055   \bbl@replace\bbl@toreplace{[d]}{\BabelDated{####3}}%
2056   \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{####3}}%
2057   \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{####2}}%
2058   \bbl@replace\bbl@toreplace{[MM]}{\BabelDateMM{####2}}%
2059   \bbl@replace\bbl@toreplace{[MMMM]}{\BabelDateMMMM{####2}}%
2060   \bbl@replace\bbl@toreplace{[y]}{\BabelDatey{####1}}%
2061   \bbl@replace\bbl@toreplace{[yy]}{\BabelDateyy{####1}}%
2062   \bbl@replace\bbl@toreplace{[yyyy]}{\BabelDateyyyy{####1}}%
2063 % Note after \bbl@replace \toks@ contains the resulting string.
2064 % TODO - Using this implicit behavior doesn't seem a good idea.
2065   \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.

```
2066 \def\bbl@provide@lsys#1{%
2067   \bbl@ifunset{bbl@lname@#1}%
2068     {\bbl@ini@ids{#1}}%
2069     {}%
2070   \bbl@csarg\let{lsys@#1}\@empty
2071   \bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}{}%
2072   \bbl@ifunset{bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{}%
2073   \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
2074   \bbl@ifunset{bbl@lname@#1}{}%
2075     {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
2076   \bbl@csarg\bbl@toglobal{lsys@#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.

```
2077 \def\bbl@ini@ids#1{%
2078   \def\BabelBeforeIni##1##2{%
2079     \begingroup
2080       \bbl@add\bbl@secpost@identification{\closein1 }%
2081       \catcode`\[=12 \catcode`\]=12 \catcode`\==12 %
2082       \bbl@read@ini{##1}%
2083     \endgroup}%            boxed, to avoid extra spaces:
2084   {\setbox\z@\hbox{\InputIfFileExists{babel-#1.tex}{}{}}}}
```

103

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

10.1 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`.

```
2085 {\def\format{lplain}
2086 \ifx\fmtname\format
2087 \else
2088   \def\format{LaTeX2e}
2089   \ifx\fmtname\format
2090   \else
2091     \aftergroup\endinput
2092   \fi
2093 \fi}
```

10.2 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.

```
2094 %\bbl@redefine\newlabel#1#2{%
2095 %   \@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.

```
2096 ⟨∗More package options⟩ ≡
2097 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
2098 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
2099 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
2100 ⟨⟨/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.

```
2101 \bbl@trace{Cross referencing macros}
2102 \ifx\bbl@opt@safe\@empty\else
2103   \def\@newl@bel#1#2#3{%
2104    {\@safe@activestrue
2105     \bbl@ifunset{#1@#2}%
2106        \relax
2107        {\gdef\@multiplelabels{%
2108           \@latex@warning@no@line{There were multiply-defined labels}}%
2109         \@latex@warning@no@line{Label `#2' multiply defined}}%
2110     \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.

```
2111  \CheckCommand*\@testdef[3]{%
2112    \def\reserved@a{#3}%
2113    \expandafter\ifx\csname#1@#2\endcsname\reserved@a
2114    \else
2115      \@tempswatrue
2116    \fi}
```

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

```
2117  \def\@testdef#1#2#3{%
2118    \@safe@activestrue
```

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

```
2119    \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
```

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

```
2120    \def\bbl@tempb{#3}%
2121    \@safe@activesfalse
```

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

```
2122    \ifx\bbl@tempa\relax
2123    \else
2124      \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
2125    \fi
```

We do the same for \bbl@tempb.

```
2126    \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
```

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

```
2127    \ifx\bbl@tempa\bbl@tempb
2128    \else
2129      \@tempswatrue
2130    \fi}
2131 \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.

```
2132 \bbl@xin@{R}\bbl@opt@safe
2133 \ifin@
```

```
2134  \bbl@redefinerobust\ref#1{%
2135     \@safe@activestrue\org@ref{#1}\@safe@activesfalse}
2136  \bbl@redefinerobust\pageref#1{%
2137     \@safe@activestrue\org@pageref{#1}\@safe@activesfalse}
2138 \else
2139    \let\org@ref\ref
2140    \let\org@pageref\pageref
2141 \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.

```
2142 \bbl@xin@{B}\bbl@opt@safe
2143 \ifin@
2144    \bbl@redefine\@citex[#1]#2{%
2145       \@safe@activestrue\edef\@tempa{#2}\@safe@activesfalse
2146       \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.

```
2147    \AtBeginDocument{%
2148       \@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.)

```
2149       \def\@citex[#1][#2]#3{%
2150          \@safe@activestrue\edef\@tempa{#3}\@safe@activesfalse
2151          \org@@citex[#1][#2]{\@tempa}}%
2152       }{}}
```

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

```
2153    \AtBeginDocument{%
2154       \@ifpackageloaded{cite}{%
2155          \def\@citex[#1]#2{%
2156             \@safe@activestrue\org@@citex[#1]{#2}\@safe@activesfalse}%
2157          }{}}
```

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

```
2158    \bbl@redefine\nocite#1{%
2159       \@safe@activestrue\org@nocite{#1}\@safe@activesfalse}
```

\bibcite The macro that is used in the .aux file to define citation labels. When packages such as natbib or cite are not loaded its second argument is used to typeset the citation label. In that case, this second argument can contain active characters but is used in an environment where \@safe@activestrue is in effect. This switch needs to be reset inside the \hbox which contains the citation label. In order to determine during .aux file processing which definition of \bibcite is needed we define \bibcite in such a way that it redefines itself with the proper definition. We call \bbl@cite@choice to select the proper definition for \bibcite. This new definition is then activated.

106

```
2160    \bbl@redefine\bibcite{%
2161      \bbl@cite@choice
2162      \bibcite}
```

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

```
2163    \def\bbl@bibcite#1#2{%
2164      \org@bibcite{#1}{\@safe@activesfalse#2}}
```

\bbl@cite@choice   The macro \bbl@cite@choice determines which definition of \bibcite is needed. First we give \bibcite its default definition.

```
2165    \def\bbl@cite@choice{%
2166      \global\let\bibcite\bbl@bibcite
```

Then, when natbib is loaded we restore the original definition of \bibcite. For cite we do the same.

```
2167      \@ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{}%
2168      \@ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{}%
```

Make sure this only happens once.

```
2169      \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.

```
2170    \AtBeginDocument{\bbl@cite@choice}
```

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

```
2171    \bbl@redefine\@bibitem#1{%
2172      \@safe@activestrue\org@@bibitem{#1}\@safe@activesfalse}
2173 \else
2174    \let\org@nocite\nocite
2175    \let\org@@citex\@citex
2176    \let\org@bibcite\bibcite
2177    \let\org@@bibitem\@bibitem
2178 \fi
```

10.3   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.

```
2179 \bbl@trace{Marks}
2180 \IfBabelLayout{sectioning}
2181    {\ifx\bbl@opt@headfoot\@nnil
2182      \g@addto@macro\@resetactivechars{%
2183        \set@typeset@protect
2184        \expandafter\select@language@x\expandafter{\bbl@main@language}%
2185        \let\protect\noexpand
2186        \edef\thepage{%
```

```
2187            \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}}%
2188      \fi}
2189    {\bbl@redefine\markright#1{%
2190       \bbl@ifblank{#1}%
2191         {\org@markright{}}%
2192         {\toks@{#1}%
2193          \bbl@exp{%
2194            \\\org@markright{\\\protect\\\foreignlanguage{\languagename}%
2195               {\\\protect\\\bbl@restore@actives\the\toks@}}}}}%
```

\markboth   The definition of \markboth is equivalent to that of \markright, except that we need two
\@mkboth   token registers. The documentclasses report and book define and set the headings for the
            page. While doing so they also store a copy of \markboth in \@mkboth. Therefore we need
            to check whether \@mkboth has already been set. If so we neeed to do that again with the
            new definition of \markboth.

```
2196    \ifx\@mkboth\markboth
2197      \def\bbl@tempc{\let\@mkboth\markboth}
2198    \else
2199      \def\bbl@tempc{}
2200    \fi
```

Now we can start the new definition of \markboth

```
2201    \bbl@redefine\markboth#1#2{%
2202      \protected@edef\bbl@tempb##1{%
2203        \protect\foreignlanguage
2204        {\languagename}{\protect\bbl@restore@actives##1}}%
2205      \bbl@ifblank{#1}%
2206        {\toks@{}}%
2207        {\toks@\expandafter{\bbl@tempb{#1}}}%
2208      \bbl@ifblank{#2}%
2209        {\@temptokena{}}%
2210        {\@temptokena\expandafter{\bbl@tempb{#2}}}%
2211      \bbl@exp{\\\org@markboth{\the\toks@}{\the\@temptokena}}}
```

and copy it to \@mkboth if necessary.

```
2212    \bbl@tempc}  % end \IfBabelLayout
```

10.4   Preventing clashes with other packages

10.4.1 `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.

```
2213  \bbl@trace{Preventing clashes with other packages}
2214  \bbl@xin@{R}\bbl@opt@safe
2215  \ifin@
2216    \AtBeginDocument{%
2217      \@ifpackageloaded{ifthen}{%
```

108

Then we can redefine \ifthenelse:

```
2218        \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.

```
2219        \let\bbl@temp@pref\pageref
2220        \let\pageref\org@pageref
2221        \let\bbl@temp@ref\ref
2222        \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.

```
2223        \@safe@activestrue
2224        \org@ifthenelse{#1}%
2225          {\let\pageref\bbl@temp@pref
2226           \let\ref\bbl@temp@ref
2227           \@safe@activesfalse
2228           #2}%
2229          {\let\pageref\bbl@temp@pref
2230           \let\ref\bbl@temp@ref
2231           \@safe@activesfalse
2232           #3}%
2233        }%
2234      }{}%
2235    }
```

10.4.2 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.

```
2236  \AtBeginDocument{%
2237    \@ifpackageloaded{varioref}{%
2238      \bbl@redefine\@@vpageref#1[#2]#3{%
2239        \@safe@activestrue
2240        \org@@@vpageref{#1}[#2]{#3}%
2241        \@safe@activesfalse}%
```

The same needs to happen for \vrefpagenum.

```
2242      \bbl@redefine\vrefpagenum#1#2{%
2243        \@safe@activestrue
2244        \org@vrefpagenum{#1}{#2}%
2245        \@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.

```
2246      \expandafter\def\csname Ref \endcsname#1{%
2247        \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
2248      }{}%
2249    }
2250 \fi
```

10.4.3 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.

```
2251 \AtEndOfPackage{%
2252   \AtBeginDocument{%
2253     \@ifpackageloaded{hhline}%
```

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

```
2254     {\expandafter\ifx\csname normal@char\string:\endcsname\relax
2255      \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.

```
2256       \makeatletter
2257       \def\@currname{hhline}\input{hhline.sty}\makeatother
2258     \fi}%
2259   {}}}
```

10.4.4 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.

```
2260 \AtBeginDocument{%
2261   \ifx\pdfstringdefDisableCommands\@undefined\else
2262     \pdfstringdefDisableCommands{\languageshorthands{system}}%
2263   \fi}
```

10.4.5 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.

```
2264 \DeclareRobustCommand{\FOREIGNLANGUAGE}[1]{%
2265   \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.

```
2266 \def\substitutefontfamily#1#2#3{%
2267   \lowercase{\immediate\openout15=#1#2.fd\relax}%
2268   \immediate\write15{%
2269     \string\ProvidesFile{#1#2.fd}%
2270     [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
2271      \space generated font description file]^^J
2272     \string\DeclareFontFamily{#1}{#2}{}^^J
2273     \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{}^^J
2274     \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{}^^J
2275     \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{}^^J
2276     \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{}^^J
2277     \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{}^^J
```

```
2278    \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{}^^J
2279    \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{}^^J
2280    \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{}^^J
2281    }%
2282  \closeout15
2283  }
```

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

```
2284 \@onlypreamble\substitutefontfamily
```

10.5 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
```
2285 \bbl@trace{Encoding and fonts}
2286 \newcommand\BabelNonASCII{LGR,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU}
2287 \newcommand\BabelNonText{TS1,T3,TS3}
2288 \let\org@TeX\TeX
2289 \let\org@LaTeX\LaTeX
2290 \let\ensureascii\@firstofone
2291 \AtBeginDocument{%
2292   \in@false
2293   \bbl@foreach\BabelNonASCII{% is there a text non-ascii enc?
2294     \ifin@\else
2295       \lowercase{\bbl@xin@{,#1enc.def,}{,\@filelist,}}%
2296     \fi}%
2297   \ifin@ % if a text non-ascii has been loaded
2298     \def\ensureascii#1{{\fontencoding{OT1}\selectfont#1}}%
2299     \DeclareTextCommandDefault{\TeX}{\org@TeX}%
2300     \DeclareTextCommandDefault{\LaTeX}{\org@LaTeX}%
2301     \def\bbl@tempb#1\@@{\uppercase{\bbl@tempc#1}ENC.DEF\@empty\@@}%
2302     \def\bbl@tempc#1ENC.DEF#2\@@{%
2303       \ifx\@empty#2\else
2304         \bbl@ifunset{T@#1}%
2305           {}%
2306           {\bbl@xin@{,#1,}{,\BabelNonASCII,\BabelNonText,}%
2307            \ifin@
2308              \DeclareTextCommand{\TeX}{#1}{\ensureascii{\org@TeX}}%
2309              \DeclareTextCommand{\LaTeX}{#1}{\ensureascii{\org@LaTeX}}%
2310            \else
2311              \def\ensureascii##1{{\fontencoding{#1}\selectfont##1}}%
2312            \fi}%
2313       \fi}%
2314     \bbl@foreach\@filelist{\bbl@tempb#1\@@}%  TODO - \@@ de mas??
2315     \bbl@xin@{,\cf@encoding,}{,\BabelNonASCII,\BabelNonText,}%
2316     \ifin@\else
2317       \edef\ensureascii#1{{%
2318         \noexpand\fontencoding{\cf@encoding}\noexpand\selectfont#1}}%
2319     \fi
2320   \fi}
```

111

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.

```
2321 \AtEndOfPackage{\edef\latinencoding{\cf@encoding}}
```

But this might be overruled with a later loading of the package fontenc. Therefore we check at the execution of \begin{document} whether it was loaded with the T1 option. The normal way to do this (using \@ifpackageloaded) is disabled for this package. Now we have to revert to parsing the internal macro \@filelist which contains all the filenames loaded.

```
2322 \AtBeginDocument{%
2323   \@ifpackageloaded{fontspec}%
2324     {\xdef\latinencoding{%
2325        \ifx\UTFencname\@undefined
2326          EU\ifcase\bbl@engine\or2\or1\fi
2327        \else
2328          \UTFencname
2329        \fi}}%
2330     {\gdef\latinencoding{OT1}%
2331      \ifx\cf@encoding\bbl@t@one
2332        \xdef\latinencoding{\bbl@t@one}%
2333      \else
2334        \@ifl@aded{def}{t1enc}{\xdef\latinencoding{\bbl@t@one}}{}%
2335      \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.

```
2336 \DeclareRobustCommand{\latintext}{%
2337   \fontencoding{\latinencoding}\selectfont
2338   \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.

```
2339 \ifx\@undefined\DeclareTextFontCommand
2340   \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
2341 \else
2342   \DeclareTextFontCommand{\textlatin}{\latintext}
2343 \fi
```

10.6   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.

112

- 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, 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>).

```
2344 \bbl@trace{Basic (internal) bidi support}
2345 \def\bbl@alscripts{,Arabic,Syriac,Thaana,}
2346 \def\bbl@rscripts{%
2347   ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
2348   Old Hungarian,Old Hungarian,Lydian,Mandaean,Manichaean,%
2349   Manichaean,Meroitic Cursive,Meroitic,Old North Arabian,%
2350   Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
2351   Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
2352   Old South Arabian,}%
2353 \def\bbl@provide@dirs#1{%
2354   \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
2355   \ifin@
2356     \global\bbl@csarg\chardef{wdir@#1}\@ne
2357     \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
2358     \ifin@
2359       \global\bbl@csarg\chardef{wdir@#1}\tw@  % useless in xetex
2360     \fi
2361   \else
2362     \global\bbl@csarg\chardef{wdir@#1}\z@
2363   \fi}
2364 \def\bbl@switchdir{%
2365   \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
2366   \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
2367   \bbl@exp{\\\bbl@setdirs\bbl@cs{wdir@\languagename}}}
2368 \def\bbl@setdirs#1{% TODO - math
2369   \ifcase\bbl@select@type % TODO - strictly, not the right test
2370     \bbl@bodydir{#1}%
2371     \bbl@pardir{#1}%
2372   \fi
2373   \bbl@textdir{#1}}
2374 \ifodd\bbl@engine  % luatex=1
2375   \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
2376   \DisableBabelHook{babel-bidi}
2377   \chardef\bbl@thetextdir\z@
2378   \chardef\bbl@thepardir\z@
2379   \def\bbl@getluadir#1{%
2380     \directlua{
2381       if tex.#1dir == 'TLT' then
2382         tex.sprint('0')
2383       elseif tex.#1dir == 'TRT' then
2384         tex.sprint('1')
2385       end}}
2386   \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
2387     \ifcase#3\relax
2388       \ifcase\bbl@getluadir{#1}\relax\else
2389         #2 TLT\relax
2390       \fi
```

113

```
2391        \else
2392          \ifcase\bbl@getluadir{#1}\relax
2393            #2 TRT\relax
2394          \fi
2395        \fi}
2396      \def\bbl@textdir#1{%
2397        \bbl@setluadir{text}\textdir{#1}% TODO - ?\linedir
2398        \chardef\bbl@thetextdir#1\relax
2399        \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*3+#1}}
2400      \def\bbl@pardir#1{%
2401        \bbl@setluadir{par}\pardir{#1}%
2402        \chardef\bbl@thepardir#1\relax}
2403      \def\bbl@bodydir{\bbl@setluadir{body}\bodydir}
2404      \def\bbl@pagedir{\bbl@setluadir{page}\pagedir}
2405      \def\bbl@dirparastext{\pardir\the\textdir\relax}%    %%%%
2406      % Sadly, we have to deal with boxes in math with basic.
2407      % Activated every math with the package option bidi=:
2408      \def\bbl@mathboxdir{%
2409        \ifcase\bbl@thetextdir\relax
2410          \everyhbox{\bgroup\aftergroup\egroup\textdir TLT\relax}%
2411        \else
2412          \everyhbox{\bgroup\aftergroup\egroup\textdir TRT\relax}%
2413        \fi}
2414    \else % pdftex=0, xetex=2
2415      \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
2416      \DisableBabelHook{babel-bidi}
2417      \newcount\bbl@dirlevel
2418      \chardef\bbl@thetextdir\z@
2419      \chardef\bbl@thepardir\z@
2420      \def\bbl@textdir#1{%
2421        \ifcase#1\relax
2422          \chardef\bbl@thetextdir\z@
2423          \bbl@textdir@i\beginL\endL
2424        \else
2425          \chardef\bbl@thetextdir\@ne
2426          \bbl@textdir@i\beginR\endR
2427        \fi}
2428      \def\bbl@textdir@i#1#2{%
2429        \ifhmode
2430          \ifnum\currentgrouplevel>\z@
2431            \ifnum\currentgrouplevel=\bbl@dirlevel
2432              \bbl@error{Multiple bidi settings inside a group}%
2433                {I'll insert a new group, but expect wrong results.}%
2434              \bgroup\aftergroup#2\aftergroup\egroup
2435            \else
2436              \ifcase\currentgrouptype\or % 0 bottom
2437                \aftergroup#2% 1 simple {}
2438              \or
2439                \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
2440              \or
2441                \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
2442              \or\or\or % vbox vtop align
2443              \or
2444                \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
2445              \or\or\or\or\or\or % output math disc insert vcent mathchoice
2446              \or
2447                \aftergroup#2% 14 \begingroup
2448              \else
2449                \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
```

```
2450           \fi
2451         \fi
2452         \bbl@dirlevel\currentgrouplevel
2453       \fi
2454       #1%
2455     \fi}
2456   \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
2457   \let\bbl@bodydir\@gobble
2458   \let\bbl@pagedir\@gobble
2459   \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).

```
2460   \def\bbl@xebidipar{%
2461     \let\bbl@xebidipar\relax
2462     \TeXXeTstate\@ne
2463     \def\bbl@xeeverypar{%
2464       \ifcase\bbl@thepardir
2465         \ifcase\bbl@thetextdir\else\beginR\fi
2466       \else
2467         {\setbox\z@\lastbox\beginR\box\z@}%
2468       \fi}%
2469     \let\bbl@severypar\everypar
2470     \newtoks\everypar
2471     \everypar=\bbl@severypar
2472     \bbl@severypar{\bbl@xeeverypar\the\everypar}}
2473   \@ifpackagewith{babel}{bidi=bidi}%
2474     {\let\bbl@textdir@i\@gobbletwo
2475      \let\bbl@xebidipar\@empty
2476      \AddBabelHook{bidi}{foreign}{%
2477        \def\bbl@tempa{\def\BabelText####1}%
2478        \ifcase\bbl@thetextdir
2479          \expandafter\bbl@tempa\expandafter{\BabelText{\LR{##1}}}%
2480        \else
2481          \expandafter\bbl@tempa\expandafter{\BabelText{\RL{##1}}}%
2482        \fi}
2483      \def\bbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}}
2484     {}%
2485 \fi
```

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

```
2486 \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}
2487 \AtBeginDocument{%
2488   \ifx\pdfstringdefDisableCommands\@undefined\else
2489     \ifx\pdfstringdefDisableCommands\relax\else
2490       \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
2491     \fi
2492   \fi}
```

10.7 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.

115

```
2493 \bbl@trace{Local Language Configuration}
2494 \ifx\loadlocalcfg\@undefined
2495   \@ifpackagewith{babel}{noconfigs}%
2496     {\let\loadlocalcfg\@gobble}%
2497     {\def\loadlocalcfg#1{%
2498       \InputIfFileExists{#1.cfg}%
2499         {\typeout{************************************^^J%
2500                   * Local config file #1.cfg used^^J%
2501                   *}}%
2502       \@empty}}
2503 \fi
```

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

```
2504 \ifx\@unexpandable@protect\@undefined
2505   \def\@unexpandable@protect{\noexpand\protect\noexpand}
2506   \long\def\protected@write#1#2#3{%
2507     \begingroup
2508       \let\thepage\relax
2509       #2%
2510       \let\protect\@unexpandable@protect
2511       \edef\reserved@a{\write#1{#3}}%
2512       \reserved@a
2513     \endgroup
2514     \if@nobreak\ifvmode\nobreak\fi\fi}
2515 \fi
2516 ⟨/core⟩
2517 ⟨*kernel⟩
```

11   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.

```
2518 ⟨⟨Make sure ProvidesFile is defined⟩⟩
2519 \ProvidesFile{switch.def}[⟨⟨date⟩⟩ ⟨⟨version⟩⟩ Babel switching mechanism]
2520 ⟨⟨Load macros for plain if not LaTeX⟩⟩
2521 ⟨⟨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.

```
2522 \def\bbl@version{⟨⟨version⟩⟩}
2523 \def\bbl@date{⟨⟨date⟩⟩}
2524 \def\adddialect#1#2{%
2525   \global\chardef#1#2\relax
2526   \bbl@usehooks{adddialect}{{#1}{#2}}%
2527   \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.

```
2528 \def\bbl@fixname#1{%
2529   \begingroup
```

```
2530     \def\bbl@tempe{l@}%
2531     \edef\bbl@tempd{\noexpand\@ifundefined{\noexpand\bbl@tempe#1}}%
2532     \bbl@tempd
2533       {\lowercase\expandafter{\bbl@tempd}%
2534         {\uppercase\expandafter{\bbl@tempd}%
2535           \@empty
2536           {\edef\bbl@tempd{\def\noexpand#1{#1}}%
2537             \uppercase\expandafter{\bbl@tempd}}}%
2538         {\edef\bbl@tempd{\def\noexpand#1{#1}}%
2539           \lowercase\expandafter{\bbl@tempd}}}%
2540       \@empty
2541     \edef\bbl@tempd{\endgroup\def\noexpand#1{#1}}%
2542   \bbl@tempd}
2543 \def\bbl@iflanguage#1{%
2544   \@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.

```
2545 \def\iflanguage#1{%
2546   \bbl@iflanguage{#1}{%
2547     \ifnum\csname l@#1\endcsname=\language
2548       \expandafter\@firstoftwo
2549     \else
2550       \expandafter\@secondoftwo
2551     \fi}}
```

11.1   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.

```
2552 \let\bbl@select@type\z@
2553 \edef\selectlanguage{%
2554   \noexpand\protect
2555   \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`.

```
2556 \ifx\@undefined\protect\let\protect\relax\fi
```

117

As LATEX 2.09 writes to files *expanded* whereas LATEX 2$_\varepsilon$ 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.

```
2557 \ifx\documentclass\@undefined
2558   \def\xstring{\string\string\string}
2559 \else
2560   \let\xstring\string
2561 \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.

```
2562 \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:

```
2563 \def\bbl@push@language{%
2564   \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.

```
2565 \def\bbl@pop@lang#1+#2-#3{%
2566   \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.

```
2567 \let\bbl@ifrestoring\@secondoftwo
2568 \def\bbl@pop@language{%
2569   \expandafter\bbl@pop@lang\bbl@language@stack-\bbl@language@stack
2570   \let\bbl@ifrestoring\@firstoftwo
2571   \expandafter\bbl@set@language\expandafter{\languagename}%
2572   \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.

118

```
2573 \expandafter\def\csname selectlanguage \endcsname#1{%
2574   \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\tw@\fi
2575   \bbl@push@language
2576   \aftergroup\bbl@pop@language
2577   \bbl@set@language{#1}}
```

\bbl@set@language   The macro \bbl@set@language takes care of switching the language environment *and* of
writing entries on the auxiliary files. For historial reasons, language names can be either
language of \language. To catch either form a trick is used, but unfortunately as a side
effect the catcodes of letters in \languagename are messed up. This is a bug, but preserved
for backwards compatibility. The list of auxiliary files can be extended by redefining
\BabelContentsFiles, but make sure they are loaded inside a group (as aux, toc, lof, and
lot do) or the last language of the document will remain active afterwards.
We also write a command to change the current language in the auxiliary files.

```
2578 \def\BabelContentsFiles{toc,lof,lot}
2579 \def\bbl@set@language#1{% from selectlanguage, pop@
2580   \edef\languagename{%
2581     \ifnum\escapechar=\expandafter`\string#1\@empty
2582     \else\string#1\@empty\fi}%
2583   \select@language{\languagename}%
2584   % write to auxs
2585   \expandafter\ifx\csname date\languagename\endcsname\relax\else
2586     \if@filesw
2587       \protected@write\@auxout{}{\string\babel@aux{\languagename}{}}%
2588       \bbl@usehooks{write}{}%
2589     \fi
2590   \fi}
2591 \def\select@language#1{% from set@, babel@aux
2592   % set hymap
2593   \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
2594   % set name
2595   \edef\languagename{#1}%
2596   \bbl@fixname\languagename
2597   \bbl@iflanguage\languagename{%
2598     \expandafter\ifx\csname date\languagename\endcsname\relax
2599       \bbl@error
2600         {Unknown language `#1'. Either you have\\%
2601          misspelled its name, it has not been installed,\\%
2602          or you requested it in a previous run. Fix its name,\\%
2603          install it or just rerun the file, respectively. In\\%
2604          some cases, you may need to remove the aux file}%
2605        {You may proceed, but expect wrong results}%
2606     \else
2607       % set type
2608       \let\bbl@select@type\z@
2609       \expandafter\bbl@switch\expandafter{\languagename}%
2610     \fi}}
2611 \def\babel@aux#1#2{%
2612   \expandafter\ifx\csname date#1\endcsname\relax
2613     \expandafter\ifx\csname bbl@auxwarn@#1\endcsname\relax
2614       \@namedef{bbl@auxwarn@#1}{}%
2615       \bbl@warning
2616         {Unknown language `#1'. Very likely you\\%
2617          requested it in a previous run. Expect some\\%
2618          wrong results in this run, which should vanish\\%
2619          in the next one. Reported}%
2620     \fi
2621   \else
```

119

```
2622      \select@language{#1}%
2623      \bbl@foreach\BabelContentsFiles{%
2624        \@writefile{##1}{\babel@toc{#1}{#2}}}}% %% TODO - ok in plain?
2625    \fi}
2626 \def\babel@toc#1#2{%
2627    \select@language{#1}}
```

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

```
2628 \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.

```
2629 \newif\ifbbl@usedategroup
2630 \def\bbl@switch#1{%  from select@, foreign@
2631   % restore
2632   \originalTeX
2633   \expandafter\def\expandafter\originalTeX\expandafter{%
2634     \csname noextras#1\endcsname
2635     \let\originalTeX\@empty
2636     \babel@beginsave}%
2637   \bbl@usehooks{afterreset}{}%
2638   \languageshorthands{none}%
2639   % switch captions, date
2640   \ifcase\bbl@select@type
2641     \ifhmode
2642       \hskip\z@skip % trick to ignore spaces
2643       \csname captions#1\endcsname\relax
2644       \csname date#1\endcsname\relax
2645       \loop\ifdim\lastskip>\z@\unskip\repeat\unskip
2646     \else
2647       \csname captions#1\endcsname\relax
2648       \csname date#1\endcsname\relax
2649     \fi
2650   \else
2651     \ifbbl@usedategroup   % if \foreign... within \<lang>date
2652       \bbl@usedategroupfalse
2653       \ifhmode
2654         \hskip\z@skip % trick to ignore spaces
2655         \csname date#1\endcsname\relax
2656         \loop\ifdim\lastskip>\z@\unskip\repeat\unskip
2657       \else
2658         \csname date#1\endcsname\relax
2659       \fi
2660     \fi
```

```
2661    \fi
2662    % switch extras
2663    \bbl@usehooks{beforeextras}{}%
2664    \csname extras#1\endcsname\relax
2665    \bbl@usehooks{afterextras}{}%
2666    %  > babel-ensure
2667    %  > babel-sh-<short>
2668    %  > babel-bidi
2669    %  > babel-fontspec
2670    % hyphenation - case mapping
2671    \ifcase\bbl@opt@hyphenmap\or
2672      \def\BabelLower##1##2{\lccode##1=##2\relax}%
2673      \ifnum\bbl@hymapsel>4\else
2674        \csname\languagename @bbl@hyphenmap\endcsname
2675      \fi
2676      \chardef\bbl@opt@hyphenmap\z@
2677    \else
2678      \ifnum\bbl@hymapsel>\bbl@opt@hyphenmap\else
2679        \csname\languagename @bbl@hyphenmap\endcsname
2680      \fi
2681    \fi
2682    \global\let\bbl@hymapsel\@cclv
2683    % hyphenation - patterns
2684    \bbl@patterns{#1}%
2685    % hyphenation - mins
2686    \babel@savevariable\lefthyphenmin
2687    \babel@savevariable\righthyphenmin
2688    \expandafter\ifx\csname #1hyphenmins\endcsname\relax
2689      \set@hyphenmins\tw@\thr@@\relax
2690    \else
2691      \expandafter\expandafter\expandafter\set@hyphenmins
2692        \csname #1hyphenmins\endcsname\relax
2693    \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.

```
2694 \long\def\otherlanguage#1{%
2695    \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\thr@@\fi
2696    \csname selectlanguage \endcsname{#1}%
2697    \ignorespaces}
```

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

```
2698 \long\def\endotherlanguage{%
2699    \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.

```
2700 \expandafter\def\csname otherlanguage*\endcsname#1{%
2701    \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
2702    \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".

```
2703 \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.

```
2704 \providecommand\bbl@beforeforeign{}
2705 \edef\foreignlanguage{%
2706   \noexpand\protect
2707   \expandafter\noexpand\csname foreignlanguage \endcsname}
2708 \expandafter\def\csname foreignlanguage \endcsname{%
2709   \@ifstar\bbl@foreign@s\bbl@foreign@x}
2710 \def\bbl@foreign@x#1#2{%
2711   \begingroup
2712     \let\BabelText\@firstofone
2713     \bbl@beforeforeign
2714     \foreign@language{#1}%
2715     \bbl@usehooks{foreign}{}%
2716     \BabelText{#2}% Now in horizontal mode!
2717   \endgroup}
2718 \def\bbl@foreign@s#1#2{% TODO - \shapemode, \@setpar, ?\@@par
2719   \begingroup
2720     {\par}%
2721     \let\BabelText\@firstofone
2722     \foreign@language{#1}%
2723     \bbl@usehooks{foreign*}{}%
2724     \bbl@dirparastext
2725     \BabelText{#2}% Still in vertical mode!
2726     {\par}%
2727   \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.

```
2728 \def\foreign@language#1{%
2729   % set name
2730   \edef\languagename{#1}%
2731   \bbl@fixname\languagename
2732   \bbl@iflanguage\languagename{%
2733     \expandafter\ifx\csname date\languagename\endcsname\relax
2734       \bbl@warning    % TODO - why a warning, not an error?
2735         {Unknown language `#1'. Either you have\\%
2736          misspelled its name, it has not been installed,\\%
2737          or you requested it in a previous run. Fix its name,\\%
2738          install it or just rerun the file, respectively. In\\%
2739          some cases, you may need to remove the aux file.\\%
2740          I'll proceed, but expect wrong results.\\%
2741          Reported}%
2742     \fi
2743   % set type
2744   \let\bbl@select@type\@ne
2745   \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.

```
2746 \let\bbl@hyphlist\@empty
2747 \let\bbl@hyphenation@\relax
2748 \let\bbl@pttnlist\@empty
2749 \let\bbl@patterns@\relax
2750 \let\bbl@hymapsel=\@cclv
2751 \def\bbl@patterns#1{%
2752   \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
2753       \csname l@#1\endcsname
2754       \edef\bbl@tempa{#1}%
2755     \else
2756       \csname l@#1:\f@encoding\endcsname
2757       \edef\bbl@tempa{#1:\f@encoding}%
2758     \fi
2759   \@expandtwoargs\bbl@usehooks{patterns}{{#1}{\bbl@tempa}}%
2760   % > luatex
2761   \@ifundefined{bbl@hyphenation@}{}{% Can be \relax!
2762     \begingroup
2763       \bbl@xin@{,\number\language,}{,\bbl@hyphlist}%
2764       \ifin@\else
2765         \@expandtwoargs\bbl@usehooks{hyphenation}{{#1}{\bbl@tempa}}%
2766         \hyphenation{%
2767           \bbl@hyphenation@
2768           \@ifundefined{bbl@hyphenation@#1}%
2769             \@empty
2770             {\space\csname bbl@hyphenation@#1\endcsname}}%
2771         \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
2772       \fi
2773     \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*.

```
2774 \def\hyphenrules#1{%
2775   \edef\bbl@tempf{#1}%
2776   \bbl@fixname\bbl@tempf
2777   \bbl@iflanguage\bbl@tempf{%
2778     \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
2779     \languageshorthands{none}%
2780     \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax
2781       \set@hyphenmins\tw@\thr@@\relax
2782     \else
2783       \expandafter\expandafter\expandafter\set@hyphenmins
2784       \csname\bbl@tempf hyphenmins\endcsname\relax
2785     \fi}}
2786 \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.

```
2787 \def\providehyphenmins#1#2{%
2788   \expandafter\ifx\csname #1hyphenmins\endcsname\relax
2789     \@namedef{#1hyphenmins}{#2}%
2790   \fi}
```

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

```
2791 \def\set@hyphenmins#1#2{%
2792   \lefthyphenmin#1\relax
2793   \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.

```
2794 \ifx\ProvidesFile\@undefined
2795   \def\ProvidesLanguage#1[#2 #3 #4]{%
2796     \wlog{Language: #1 #4 #3 <#2>}%
2797     }
2798 \else
2799   \def\ProvidesLanguage#1{%
2800     \begingroup
2801       \catcode`\ 10 %
2802       \@makeother\/%
2803       \@ifnextchar[%
2804         {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}}
2805   \def\@provideslanguage#1[#2]{%
2806     \wlog{Language: #1 #2}%
2807     \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
2808     \endgroup}
2809 \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`

```
2810 \def\LdfInit{%
2811   \chardef\atcatcode=\catcode`\@
2812   \catcode`\@=11\relax
2813   \input babel.def\relax
2814   \catcode`\@=\atcatcode \let\atcatcode\relax
2815   \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`.

```
2816 \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.

```
2817 \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':

```
2818 \providecommand\setlocale{%
2819   \bbl@error
2820     {Not yet available}%
2821     {Find an armchair, sit down and wait}}
2822 \let\uselocale\setlocale
2823 \let\locale\setlocale
2824 \let\selectlocale\setlocale
2825 \let\textlocale\setlocale
2826 \let\textlanguage\setlocale
2827 \let\languagetext\setlocale
```

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

```
2828 \edef\bbl@nulllanguage{\string\language=0}
2829 \ifx\PackageError\@undefined
2830   \def\bbl@error#1#2{%
2831     \begingroup
2832       \newlinechar=`\^^J
2833       \def\\{^^J(babel) }%
2834       \errhelp{#2}\errmessage{\\#1}%
2835     \endgroup}
2836   \def\bbl@warning#1{%
2837     \begingroup
2838       \newlinechar=`\^^J
2839       \def\\{^^J(babel) }%
2840       \message{\\#1}%
```

```
2841      \endgroup}
2842   \def\bbl@info#1{%
2843      \begingroup
2844        \newlinechar=`\^^J
2845        \def\\{^^J}%
2846        \wlog{#1}%
2847      \endgroup}
2848 \else
2849   \def\bbl@error#1#2{%
2850      \begingroup
2851        \def\\{\MessageBreak}%
2852        \PackageError{babel}{#1}{#2}%
2853      \endgroup}
2854   \def\bbl@warning#1{%
2855      \begingroup
2856        \def\\{\MessageBreak}%
2857        \PackageWarning{babel}{#1}%
2858      \endgroup}
2859   \def\bbl@info#1{%
2860      \begingroup
2861        \def\\{\MessageBreak}%
2862        \PackageInfo{babel}{#1}%
2863      \endgroup}
2864 \fi
2865 \@ifpackagewith{babel}{silent}
2866   {\let\bbl@info\@gobble
2867    \let\bbl@warning\@gobble}
2868   {}
2869 \def\bbl@nocaption{\protect\bbl@nocaption@i}
2870 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
2871   \global\@namedef{#2}{\textbf{?#1?}}%
2872   \@nameuse{#2}%
2873   \bbl@warning{%
2874     \@backslashchar#2 not set. Please, define\\%
2875     it in the preamble with something like:\\%
2876     \string\renewcommand\@backslashchar#2{..}\\%
2877     Reported}}
2878 \def\bbl@tentative{\protect\bbl@tentative@i}
2879 \def\bbl@tentative@i#1{%
2880   \bbl@warning{%
2881     Some functions for '#1' are tentative.\\%
2882     They might not work as expected and their behavior\\%
2883     could change in the future.\\%
2884     Reported}}
2885 \def\@nolanerr#1{%
2886   \bbl@error
2887     {You haven't defined the language #1\space yet}%
2888     {Your command will be ignored, type <return> to proceed}}
2889 \def\@nopatterns#1{%
2890   \bbl@warning
2891     {No hyphenation patterns were preloaded for\\%
2892      the language `#1' into the format.\\%
2893      Please, configure your TeX system to add them and\\%
2894      rebuild the format. Now I will use the patterns\\%
2895      preloaded for \bbl@nulllanguage\space instead}}
2896 \let\bbl@usehooks\@gobbletwo
2897 ⟨/kernel⟩
2898 ⟨*patterns⟩
```

126

12 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.

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.

```
2899 ⟨⟨Make sure ProvidesFile is defined⟩⟩
2900 \ProvidesFile{hyphen.cfg}[⟨⟨date⟩⟩ ⟨⟨version⟩⟩ Babel hyphens]
2901 \xdef\bbl@format{\jobname}
2902 \ifx\AtBeginDocument\@undefined
2903   \def\@empty{}
2904   \let\orig@dump\dump
2905   \def\dump{%
2906     \ifx\@ztryfc\@undefined
2907     \else
2908       \toks0=\expandafter{\@preamblecmds}%
2909       \edef\@preamblecmds{\noexpand\@begindocumenthook\the\toks0}%
2910       \def\@begindocumenthook{}%
2911     \fi
2912     \let\dump\orig@dump\let\orig@dump\@undefined\dump}
2913 \fi
2914 ⟨⟨Define core switching macros⟩⟩
```

`\process@line` Each line in the file `language.dat` is processed by `\process@line` after it is read. The first thing this macro does is to check whether the line starts with =. When the first token of a

line is an =, the macro `\process@synonym` is called; otherwise the macro
`\process@language` will continue.

```
2915 \def\process@line#1#2 #3 #4 {%
2916   \ifx=#1%
2917     \process@synonym{#2}%
2918   \else
2919     \process@language{#1#2}{#3}{#4}%
2920   \fi
2921   \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.

```
2922 \toks@{}
2923 \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.

```
2924 \def\process@synonym#1{%
2925   \ifnum\last@language=\m@ne
2926     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
2927   \else
2928     \expandafter\chardef\csname l@#1\endcsname\last@language
2929     \wlog{\string\l@#1=\string\language\the\last@language}%
2930     \expandafter\let\csname #1hyphenmins\expandafter\endcsname
2931       \csname\languagename hyphenmins\endcsname
2932     \let\bbl@elt\relax
2933     \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}{}{}}%
2934   \fi}
```

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

empty and when it does not contain a space token. (Note however there is no need to save hyphenation exceptions into the format.)

\bbl@languages saves a snapshot of the loaded 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.

```
2935 \def\process@language#1#2#3{%
2936   \expandafter\addlanguage\csname l@#1\endcsname
2937   \expandafter\language\csname l@#1\endcsname
2938   \edef\languagename{#1}%
2939   \bbl@hook@everylanguage{#1}%
2940   %  > luatex
2941   \bbl@get@enc#1::\@@@
2942   \begingroup
2943     \lefthyphenmin\m@ne
2944     \bbl@hook@loadpatterns{#2}%
2945     %  > luatex
2946     \ifnum\lefthyphenmin=\m@ne
2947     \else
2948       \expandafter\xdef\csname #1hyphenmins\endcsname{%
2949         \the\lefthyphenmin\the\righthyphenmin}%
2950     \fi
2951   \endgroup
2952   \def\bbl@tempa{#3}%
2953   \ifx\bbl@tempa\@empty\else
2954     \bbl@hook@loadexceptions{#3}%
2955     %  > luatex
2956   \fi
2957   \let\bbl@elt\relax
2958   \edef\bbl@languages{%
2959     \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
2960   \ifnum\the\language=\z@
2961     \expandafter\ifx\csname #1hyphenmins\endcsname\relax
2962       \set@hyphenmins\tw@\thr@@\relax
2963     \else
2964       \expandafter\expandafter\expandafter\set@hyphenmins
2965         \csname #1hyphenmins\endcsname
2966     \fi
2967     \the\toks@
2968     \toks@{}%
2969   \fi}
```

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

```
2970 \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.

```
2971 \def\bbl@hook@everylanguage#1{}
2972 \def\bbl@hook@loadpatterns#1{\input #1\relax}
2973 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
2974 \let\bbl@hook@loadkernel\bbl@hook@loadpatterns
2975 \begingroup
2976   \def\AddBabelHook#1#2{%
2977     \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
2978       \def\next{\toks1}%
```

```
2979      \else
2980        \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname####1}%
2981      \fi
2982      \next}
2983    \ifx\directlua\@undefined
2984      \ifx\XeTeXinputencoding\@undefined\else
2985        \input xebabel.def
2986      \fi
2987    \else
2988      \input luababel.def
2989    \fi
2990    \openin1 = babel-\bbl@format.cfg
2991    \ifeof1
2992    \else
2993      \input babel-\bbl@format.cfg\relax
2994    \fi
2995    \closein1
2996  \endgroup
2997  \bbl@hook@loadkernel{switch.def}
```

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

```
2998 \openin1 = language.dat
```

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

```
2999 \def\languagename{english}%
3000 \ifeof1
3001   \message{I couldn't find the file language.dat,\space
3002           I will try the file hyphen.tex}
3003   \input hyphen.tex\relax
3004   \chardef\l@english\z@
3005 \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.

```
3006   \last@language\m@ne
```

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

```
3007   \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.

```
3008     \endlinechar\m@ne
3009     \read1 to \bbl@line
3010     \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.

```
3011     \if T\ifeof1F\fi T\relax
3012       \ifx\bbl@line\@empty\else
3013         \edef\bbl@line{\bbl@line\space\space\space}%
3014         \expandafter\process@line\bbl@line\relax
3015       \fi
3016   \repeat
```

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

```
3017  \begingroup
3018    \def\bbl@elt#1#2#3#4{%
3019      \global\language=#2\relax
3020      \gdef\languagename{#1}%
3021      \def\bbl@elt##1##2##3##4{}}%
3022    \bbl@languages
3023  \endgroup
3024 \fi
```

and close the configuration file.

```
3025 \closein1
```

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

```
3026 \if/\the\toks@/\else
3027   \errhelp{language.dat loads no language, only synonyms}
3028   \errmessage{Orphan language synonym}
3029 \fi
```

Also remove some macros from memory and raise an error if `\toks@` is not empty. Finally load `switch.def`, but the latter is not required and the line inputting it may be commented out.

```
3030 \let\bbl@line\@undefined
3031 \let\process@line\@undefined
3032 \let\process@synonym\@undefined
3033 \let\process@language\@undefined
3034 \let\bbl@get@enc\@undefined
3035 \let\bbl@hyph@enc\@undefined
3036 \let\bbl@tempa\@undefined
3037 \let\bbl@hook@loadkernel\@undefined
3038 \let\bbl@hook@everylanguage\@undefined
3039 \let\bbl@hook@loadpatterns\@undefined
3040 \let\bbl@hook@loadexceptions\@undefined
3041 ⟨/patterns⟩
```

Here the code for iniTeX ends.

13   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].

```
3042 ⟨⟨*More package options⟩⟩ ≡
3043 \ifodd\bbl@engine
3044   \DeclareOption{bidi=basic-r}%
3045     {\ExecuteOptions{bidi=basic}}
3046   \DeclareOption{bidi=basic}%
3047     {\let\bbl@beforeforeign\leavevmode
3048      \newattribute\bbl@attr@dir
3049      % TODO - the same trick as bbl@severypar
3050      % \everymath{\bbl@mathboxdir}%
3051      % \everydisplay{\bbl@mathboxdir}%
3052      \bbl@exp{\output{\bodydir\pagedir\the\output}}%
3053      \AtEndOfPackage{\EnableBabelHook{babel-bidi}}}}
3054 \else
```

```
3055    \DeclareOption{bidi=basic-r}%
3056      {\ExecuteOptions{bidi=basic}}
3057    \DeclareOption{bidi=basic}%
3058      {\bbl@error
3059        {The bidi method `basic' is available only in\\%
3060         luatex. I'll continue with `bidi=default', so\\%
3061         expect wrong results}%
3062        {See the manual for further details.}%
3063      \let\bbl@beforeforeign\leavevmode
3064      \AtEndOfPackage{%
3065        \EnableBabelHook{babel-bidi}%
3066        \bbl@xebidipar}}
3067    \def\bbl@loadxebidi#1{%
3068      \ifx\RTLfootnotetext\@undefined
3069        \AtEndOfPackage{%
3070          \EnableBabelHook{babel-bidi}%
3071          \ifx\fontspec\@undefined
3072            \usepackage{fontspec}% bidi needs fontspec
3073          \fi
3074          \usepackage#1{bidi}}%
3075      \fi}
3076    \DeclareOption{bidi=bidi}%
3077      {\bbl@tentative{bidi=bidi}%
3078        \bbl@loadxebidi{}}
3079    \DeclareOption{bidi=bidi-r}%
3080      {\bbl@tentative{bidi=bidi-r}%
3081        \bbl@loadxebidi{[rldocument]}}
3082    \DeclareOption{bidi=bidi-l}%
3083      {\bbl@tentative{bidi=bidi-l}%
3084        \bbl@loadxebidi{}}
3085 \fi
3086 \DeclareOption{bidi=default}%
3087    {\let\bbl@beforeforeign\leavevmode
3088      \ifodd\bbl@engine
3089        \newattribute\bbl@attr@dir
3090        \bbl@exp{\output{\bodydir\pagedir\the\output}}}%
3091      \fi
3092      \AtEndOfPackage{%
3093        \EnableBabelHook{babel-bidi}%
3094        \ifodd\bbl@engine\else
3095          \bbl@xebidipar
3096      \fi}}
3097 ⟨⟨/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.

```
3098 ⟨⟨∗Font selection⟩⟩ ≡
3099 \bbl@trace{Font handling with fontspec}
3100 \@onlypreamble\babelfont
3101 \newcommand\babelfont[2][]{%  1=langs/scripts 2=fam
3102    \edef\bbl@tempa{#1}%
3103    \def\bbl@tempb{#2}%
3104    \ifx\fontspec\@undefined
3105      \usepackage{fontspec}%
3106    \fi
3107    \EnableBabelHook{babel-fontspec}% Just calls \bbl@switchfont
3108    \bbl@bblfont}
3109 \newcommand\bbl@bblfont[2][]{% 1=features 2=fontname
3110    \bbl@ifunset{\bbl@tempb family}{\bbl@providefam{\bbl@tempb}}{}%
```

```
3111  % For the default font, just in case:
3112  \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
3113  \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
3114    {\bbl@csarg\edef{\bbl@tempb dflt@}{<>{#1}{#2}}% save bbl@rmdflt@
3115     \bbl@exp{%
3116       \let\<bbl@\bbl@tempb dflt@\languagename>\<bbl@\bbl@tempb dflt@>%
3117       \\\bbl@font@set\<bbl@\bbl@tempb dflt@\languagename>%
3118                      \<\bbl@tempb default>\<\bbl@tempb family>}}%
3119    {\bbl@foreach\bbl@tempa{% ie bbl@rmdflt@lang / *scrt
3120       \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:

```
3121  \def\bbl@providefam#1{%
3122    \bbl@exp{%
3123      \\\newcommand\<#1default>{}% Just define it
3124      \\\bbl@add@list\\\bbl@font@fams{#1}%
3125      \\\DeclareRobustCommand\<#1family>{%
3126        \\\not@math@alphabet\<#1family>\relax
3127        \\\fontfamily\<#1default>\\\selectfont}%
3128      \\\DeclareTextFontCommand{\<text#1>}{\<#1family>}}}
```

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

```
3129  \def\bbl@switchfont{%
3130    \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
3131    \bbl@exp{%  eg Arabic -> arabic
3132      \lowercase{\edef\\\bbl@tempa{\bbl@cs{sname@\languagename}}}}%
3133    \bbl@foreach\bbl@font@fams{%
3134      \bbl@ifunset{bbl@##1dflt@\languagename}%    (1) language?
3135        {\bbl@ifunset{bbl@##1dflt@*\bbl@tempa}%   (2) from script?
3136           {\bbl@ifunset{bbl@##1dflt@}%           2=F - (3) from generic?
3137              {}%                                 123=F - nothing!
3138              {\bbl@exp{%                         3=T - from generic
3139                 \global\let\<bbl@##1dflt@\languagename>%
3140                            \<bbl@##1dflt@>}}}%
3141           {\bbl@exp{%                            2=T - from script
3142              \global\let\<bbl@##1dflt@\languagename>%
3143                         \<bbl@##1dflt@*\bbl@tempa>}}}%
3144        {}}%                                      1=T - language, already defined
3145    \def\bbl@tempa{%
3146      \bbl@warning{The current font is not a standard family:\\%
3147        \fontname\font\\%
3148        Script and Language are not applied. Consider\\%
3149        defining a new family with \string\babelfont.\\%
3150        Reported}}%
3151    \bbl@foreach\bbl@font@fams{%     don't gather with prev for
3152      \bbl@ifunset{bbl@##1dflt@\languagename}%
3153        {\bbl@cs{famrst@##1}%
3154         \global\bbl@csarg\let{famrst@##1}\relax}%
3155        {\bbl@exp{% order is relevant
3156           \\\bbl@add\\\originalTeX{%
3157             \\\bbl@font@rst{\bbl@cs{##1dflt@\languagename}}%
3158                          \<##1default>\<##1family>{##1}}%
3159           \\\bbl@font@set\<bbl@##1dflt@\languagename>% the main part!
3160                          \<##1default>\<##1family>}}}%
3161    \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. We must deactivate

temporarily \bbl@mapselect because \selectfont is called internally when a font is
defined.

```
3162 \def\bbl@font@set#1#2#3{% eg \bbl@rmdflt@lang \rmdefault \rmfamily
3163   \bbl@xin@{<>}{#1}%
3164   \ifin@
3165     \bbl@exp{\\\bbl@fontspec@set\\#1\expandafter\@gobbletwo#1\\#3}%
3166   \fi
3167   \bbl@exp{%
3168     \def\\#2{#1}%           eg, \rmdefault{\bbl@rmdflt@lang}
3169     \\\bbl@ifsamestring{#2}{\f@family}{\\#3\let\\\bbl@tempa\relax}{}}}
3170 %     TODO - next should be global?, but even local does its job. I'm
3171 %     still not sure -- must investigate:
3172 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
3173   \let\bbl@tempe\bbl@mapselect
3174   \let\bbl@mapselect\relax
3175   \let\bbl@temp@fam#4%       eg, '\rmfamily', to be restored below
3176   \let#4\relax            % So that can be used with \newfontfamily
3177   \bbl@exp{%
3178     \let\\\bbl@temp@pfam\<\bbl@stripslash#4\space>% eg, '\rmfamily '
3179     \<keys_if_exist:nnF>{fontspec-opentype}%
3180       {Script/\bbl@cs{sname@\languagename}}%
3181     {\\\newfontscript{\bbl@cs{sname@\languagename}}%
3182       {\bbl@cs{sotf@\languagename}}}%
3183     \<keys_if_exist:nnF>{fontspec-opentype}%
3184       {Language/\bbl@cs{lname@\languagename}}%
3185     {\\\newfontlanguage{\bbl@cs{lname@\languagename}}%
3186       {\bbl@cs{lotf@\languagename}}}%
3187     \\\newfontfamily\\#4%
3188       [\bbl@cs{lsys@\languagename},#2]}{#3}% ie \bbl@exp{..}{#3}
3189   \begingroup
3190     #4%
3191     \xdef#1{\f@family}%     eg, \bbl@rmdflt@lang{FreeSerif(0)}
3192   \endgroup
3193   \let#4\bbl@temp@fam
3194   \bbl@exp{\let\<\bbl@stripslash#4\space>}\bbl@temp@pfam
3195   \let\bbl@mapselect\bbl@tempe}%
```

font@rst and famrst are only used when there is no global settings, to save and restore de
previous families. Not really necessary, but done for optimization.

```
3196 \def\bbl@font@rst#1#2#3#4{%
3197   \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.

```
3198 \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 :-).

```
3199 \newcommand\babelFSstore[2][]{%
3200   \bbl@ifblank{#1}%
3201     {\bbl@csarg\def{sname@#2}{Latin}}%
3202     {\bbl@csarg\def{sname@#2}{#1}}%
3203   \bbl@provide@dirs{#2}%
3204   \bbl@csarg\ifnum{wdir@#2}>\z@
3205     \let\bbl@beforeforeign\leavevmode
3206     \EnableBabelHook{babel-bidi}%
3207   \fi
```

```
3208    \bbl@foreach{#2}{%
3209      \bbl@FSstore{##1}{rm}\rmdefault\bbl@save@rmdefault
3210      \bbl@FSstore{##1}{sf}\sfdefault\bbl@save@sfdefault
3211      \bbl@FSstore{##1}{tt}\ttdefault\bbl@save@ttdefault}}
3212 \def\bbl@FSstore#1#2#3#4{%
3213    \bbl@csarg\edef{#2default#1}{#3}%
3214    \expandafter\addto\csname extras#1\endcsname{%
3215      \let#4#3%
3216      \ifx#3\f@family
3217        \edef#3{\csname bbl@#2default#1\endcsname}%
3218        \fontfamily{#3}\selectfont
3219      \else
3220        \edef#3{\csname bbl@#2default#1\endcsname}%
3221      \fi}%
3222    \expandafter\addto\csname noextras#1\endcsname{%
3223      \ifx#3\f@family
3224        \fontfamily{#4}\selectfont
3225      \fi
3226      \let#3#4}}
3227 \let\bbl@langfeatures\@empty
3228 \def\babelFSfeatures{% make sure \fontspec is redefined once
3229    \let\bbl@ori@fontspec\fontspec
3230    \renewcommand\fontspec[1][]{%
3231      \bbl@ori@fontspec[\bbl@langfeatures##1]}
3232    \let\babelFSfeatures\bbl@FSfeatures
3233    \babelFSfeatures}
3234 \def\bbl@FSfeatures#1#2{%
3235    \expandafter\addto\csname extras#1\endcsname{%
3236      \babel@save\bbl@langfeatures
3237      \edef\bbl@langfeatures{#2,}}}
3238 ⟨⟨/Font selection⟩⟩
```

14   Hooks for XeTeX and LuaTeX

14.1   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.

```
3239 ⟨⟨*Restore Unicode catcodes before loading patterns⟩⟩ ≡
3240    \begingroup
3241      % Reset chars "80-"C0 to category "other", no case mapping:
3242      \catcode`\@=11 \count@=128
3243      \loop\ifnum\count@<192
3244        \global\uccode\count@=0 \global\lccode\count@=0
3245        \global\catcode\count@=12 \global\sfcode\count@=1000
3246        \advance\count@ by 1 \repeat
3247      % Other:
3248      \def\O ##1 {%
3249        \global\uccode"##1=0 \global\lccode"##1=0
3250        \global\catcode"##1=12 \global\sfcode"##1=1000 }%
3251      % Letter:
3252      \def\L ##1 ##2 ##3 {\global\catcode"##1=11
3253        \global\uccode"##1="##2
```

```
3254        \global\lccode"##1="##3
3255        % Uppercase letters have sfcode=999:
3256        \ifnum"##1="##3 \else \global\sfcode"##1=999 \fi }%
3257        % Letter without case mappings:
3258      \def\l ##1 {\L ##1 ##1 ##1 }%
3259      \l 00AA
3260      \L 00B5 039C 00B5
3261      \l 00BA
3262      \O 00D7
3263      \l 00DF
3264      \O 00F7
3265      \L 00FF 0178 00FF
3266    \endgroup
3267    \input #1\relax
3268 ⟨⟨/Restore Unicode catcodes before loading patterns⟩⟩
```

Some more common code.

```
3269 ⟨⟨∗Footnote changes⟩⟩ ≡
3270 \bbl@trace{Bidi footnotes}
3271 \ifx\bbl@beforeforeign\leavevmode
3272  \def\bbl@footnote#1#2#3{%
3273    \@ifnextchar[%
3274      {\bbl@footnote@o{#1}{#2}{#3}}%
3275      {\bbl@footnote@x{#1}{#2}{#3}}}
3276  \def\bbl@footnote@x#1#2#3#4{%
3277    \bgroup
3278      \select@language@x{\bbl@main@language}%
3279      \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
3280    \egroup}
3281  \def\bbl@footnote@o#1#2#3[#4]#5{%
3282    \bgroup
3283      \select@language@x{\bbl@main@language}%
3284      \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
3285    \egroup}
3286  \def\bbl@footnotetext#1#2#3{%
3287    \@ifnextchar[%
3288      {\bbl@footnotetext@o{#1}{#2}{#3}}%
3289      {\bbl@footnotetext@x{#1}{#2}{#3}}}
3290  \def\bbl@footnotetext@x#1#2#3#4{%
3291    \bgroup
3292      \select@language@x{\bbl@main@language}%
3293      \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
3294    \egroup}
3295  \def\bbl@footnotetext@o#1#2#3[#4]#5{%
3296    \bgroup
3297      \select@language@x{\bbl@main@language}%
3298      \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
3299    \egroup}
3300  \def\BabelFootnote#1#2#3#4{%
3301    \ifx\bbl@fn@footnote\@undefined
3302      \let\bbl@fn@footnote\footnote
3303    \fi
3304    \ifx\bbl@fn@footnotetext\@undefined
3305      \let\bbl@fn@footnotetext\footnotetext
3306    \fi
3307    \bbl@ifblank{#2}%
3308      {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
3309       \@namedef{\bbl@stripslash#1text}%
3310         {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
```

```
3311        {\def#1{\bbl@exp{\\\bbl@footnote{\\\foreignlanguage{#2}}}{#3}{#4}}%
3312          \@namedef{\bbl@stripslash#1text}%
3313            {\bbl@exp{\\\bbl@footnotetext{\\\foreignlanguage{#2}}}{#3}{#4}}}}
3314 \fi
3315 ⟨⟨/Footnote changes⟩⟩
```

Now, the code.

```
3316 ⟨∗xetex⟩
3317 \def\BabelStringsDefault{unicode}
3318 \let\xebbl@stop\relax
3319 \AddBabelHook{xetex}{encodedcommands}{%
3320   \def\bbl@tempa{#1}%
3321   \ifx\bbl@tempa\@empty
3322     \XeTeXinputencoding"bytes"%
3323   \else
3324     \XeTeXinputencoding"#1"%
3325   \fi
3326   \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
3327 \AddBabelHook{xetex}{stopcommands}{%
3328   \xebbl@stop
3329   \let\xebbl@stop\relax}
3330 \def\bbl@intraspace#1 #2 #3\@@{%
3331   \bbl@csarg\gdef{xeisp@\bbl@cs{sbcp@\languagename}}%
3332     {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
3333 \def\bbl@intrapenalty#1\@@{%
3334   \bbl@csarg\gdef{xeipn@\bbl@cs{sbcp@\languagename}}%
3335     {\XeTeXlinebreakpenalty #1\relax}}
3336 \AddBabelHook{xetex}{loadkernel}{%
3337 ⟨⟨Restore Unicode catcodes before loading patterns⟩⟩}
3338 \ifx\DisableBabelHook\@undefined\endinput\fi
3339 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
3340 \DisableBabelHook{babel-fontspec}
3341 ⟨⟨Font selection⟩⟩
3342 \input txtbabel.def
3343 ⟨/xetex⟩
```

14.2 Layout

In progress.

Note elements like headlines and margins can be modified easily with packages like fancyhdr, typearea or titleps, and geometry.

\bbl@startskip and \bbl@endskip are available to package authors. Thanks to the TEX expansion mechanism the following constructs are valid: \adim\bbl@startskip, \advance\bbl@startskip\adim, \bbl@startskip\adim.

Consider txtbabel as a shorthand for *tex–xet babel*, which is the bidi model in both pdftex and xetex.

```
3344 ⟨∗texxet⟩
3345 \bbl@trace{Redefinitions for bidi layout}
3346 \def\bbl@sspre@caption{%
3347   \bbl@exp{\everyhbox{\\\bbl@textdir\bbl@cs{wdir@\bbl@main@language}}}}
3348 \ifx\bbl@opt@layout\@nnil\endinput\fi  % No layout
3349 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
3350 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
3351 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
3352   \def\@hangfrom#1{%
3353     \setbox\@tempboxa\hbox{{#1}}%
3354     \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
```

```
3355        \noindent\box\@tempboxa}
3356    \def\raggedright{%
3357        \let\\\@centercr
3358        \bbl@startskip\z@skip
3359        \@rightskip\@flushglue
3360        \bbl@endskip\@rightskip
3361        \parindent\z@
3362        \parfillskip\bbl@startskip}
3363    \def\raggedleft{%
3364        \let\\\@centercr
3365        \bbl@startskip\@flushglue
3366        \bbl@endskip\z@skip
3367        \parindent\z@
3368        \parfillskip\bbl@endskip}
3369 \fi
3370 \IfBabelLayout{lists}
3371    {\def\list#1#2{%
3372        \ifnum \@listdepth >5\relax
3373            \@toodeep
3374        \else
3375            \global\advance\@listdepth\@ne
3376        \fi
3377        \rightmargin\z@
3378        \listparindent\z@
3379        \itemindent\z@
3380        \csname @list\romannumeral\the\@listdepth\endcsname
3381        \def\@itemlabel{#1}%
3382        \let\makelabel\@mklab
3383        \@nmbrlistfalse
3384        #2\relax
3385        \@trivlist
3386        \parskip\parsep
3387        \parindent\listparindent
3388        \advance\linewidth-\rightmargin
3389        \advance\linewidth-\leftmargin
3390        \advance\@totalleftmargin
3391            \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi
3392        \parshape\@ne\@totalleftmargin\linewidth
3393        \ignorespaces}%
3394    \ifcase\bbl@engine
3395        \def\labelenumii{)\theenumii(}%
3396        \def\p@enumiii{\p@enumii)\theenumii(}%
3397    \fi
3398    \def\@verbatim{%
3399        \trivlist \item\relax
3400        \if@minipage\else\vskip\parskip\fi
3401        \bbl@startskip\textwidth
3402        \advance\bbl@startskip-\linewidth
3403        \bbl@endskip\z@skip
3404        \parindent\z@
3405        \parfillskip\@flushglue
3406        \parskip\z@skip
3407        \@@par
3408        \language\l@nohyphenation
3409        \@tempswafalse
3410        \def\par{%
3411            \if@tempswa
3412                \leavevmode\null
3413                \@@par\penalty\interlinepenalty
```

138

```
3414        \else
3415          \@tempswatrue
3416          \ifhmode\@@par\penalty\interlinepenalty\fi
3417        \fi}%
3418      \let\do\@makeother \dospecials
3419      \obeylines \verbatim@font \@noligs
3420      \everypar\expandafter{\the\everypar\unpenalty}}}
3421    {}
3422 \IfBabelLayout{contents}
3423    {\def\@dottedtocline#1#2#3#4#5{%
3424      \ifnum#1>\c@tocdepth\else
3425        \vskip \z@ \@plus.2\p@
3426        {\bbl@startskip#2\relax
3427        \bbl@endskip\@tocrmarg
3428        \parfillskip-\bbl@endskip
3429        \parindent#2\relax
3430        \@afterindenttrue
3431        \interlinepenalty\@M
3432        \leavevmode
3433        \@tempdima#3\relax
3434        \advance\bbl@startskip\@tempdima
3435        \null\nobreak\hskip-\bbl@startskip
3436        {#4}\nobreak
3437        \leaders\hbox{%
3438          $\m@th\mkern\@dotsep mu\hbox{.}\mkern\@dotsep mu$}%
3439          \hfill\nobreak
3440          \hb@xt@\@pnumwidth{\hfil\normalfont\normalcolor#5}%
3441          \par}%
3442      \fi}}
3443    {}
3444 \IfBabelLayout{columns}
3445    {\def\@outputdblcol{%
3446      \if@firstcolumn
3447        \global\@firstcolumnfalse
3448        \global\setbox\@leftcolumn\copy\@outputbox
3449        \splitmaxdepth\maxdimen
3450        \vbadness\maxdimen
3451        \setbox\@outputbox\vbox{\unvbox\@outputbox\unskip}%
3452        \setbox\@outputbox\vsplit\@outputbox to\maxdimen
3453        \toks@\expandafter{\topmark}%
3454        \xdef\@firstcoltopmark{\the\toks@}%
3455        \toks@\expandafter{\splitfirstmark}%
3456        \xdef\@firstcolfirstmark{\the\toks@}%
3457        \ifx\@firstcolfirstmark\@empty
3458          \global\let\@setmarks\relax
3459        \else
3460          \gdef\@setmarks{%
3461            \let\firstmark\@firstcolfirstmark
3462            \let\topmark\@firstcoltopmark}%
3463        \fi
3464      \else
3465        \global\@firstcolumntrue
3466        \setbox\@outputbox\vbox{%
3467          \hb@xt@\textwidth{%
3468            \hskip\columnwidth
3469            \hfil
3470            {\normalcolor\vrule \@width\columnseprule}%
3471            \hfil
3472            \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
```

```
3473              \hskip-\textwidth
3474              \hb@xt@\columnwidth{\box\@outputbox \hss}%
3475              \hskip\columnsep
3476              \hskip\columnwidth}}%
3477          \@combinedblfloats
3478          \@setmarks
3479          \@outputpage
3480          \begingroup
3481            \@dblfloatplacement
3482            \@startdblcolumn
3483            \@whilesw\if@fcolmade \fi{\@outputpage
3484            \@startdblcolumn}%
3485          \endgroup
3486      \fi}}%
3487    {}
3488 ⟨⟨Footnote changes⟩⟩
3489 \IfBabelLayout{footnotes}%
3490    {\BabelFootnote\footnote\languagename{}{}%
3491     \BabelFootnote\localfootnote\languagename{}{}%
3492     \BabelFootnote\mainfootnote{}{}{}}
3493    {}
```

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.

```
3494 \IfBabelLayout{counters}%
3495    {\let\bbl@latinarabic=\@arabic
3496     \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
3497     \let\bbl@asciiroman=\@roman
3498     \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciiroman#1}}}%
3499     \let\bbl@asciiRoman=\@Roman
3500     \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}}}{}
3501 ⟨/texxet⟩
```

14.3   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.

```
3502 ⟨∗luatex⟩
3503 \ifx\AddBabelHook\@undefined
3504 \bbl@trace{Read language.dat}
3505 \begingroup
3506   \toks@{}
3507   \count@\z@ % 0=start, 1=0th, 2=normal
3508   \def\bbl@process@line#1#2 #3 #4 {%
3509     \ifx=#1%
3510       \bbl@process@synonym{#2}%
3511     \else
3512       \bbl@process@language{#1#2}{#3}{#4}%
3513     \fi
3514     \ignorespaces}
3515   \def\bbl@manylang{%
3516     \ifnum\bbl@last>\@ne
3517       \bbl@info{Non-standard hyphenation setup}%
3518     \fi
3519     \let\bbl@manylang\relax}
3520   \def\bbl@process@language#1#2#3{%
3521     \ifcase\count@
3522       \@ifundefined{zth@#1}{\count@\tw@}{\count@\@ne}%
3523     \or
3524       \count@\tw@
3525     \fi
3526     \ifnum\count@=\tw@
3527       \expandafter\addlanguage\csname l@#1\endcsname
3528       \language\allocationnumber
3529       \chardef\bbl@last\allocationnumber
3530       \bbl@manylang
3531       \let\bbl@elt\relax
3532       \xdef\bbl@languages{%
3533         \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
3534     \fi
3535     \the\toks@
3536     \toks@{}}
3537   \def\bbl@process@synonym@aux#1#2{%
3538     \global\expandafter\chardef\csname l@#1\endcsname#2\relax
3539     \let\bbl@elt\relax
3540     \xdef\bbl@languages{%
3541       \bbl@languages\bbl@elt{#1}{#2}{}{}}%
3542   \def\bbl@process@synonym#1{%
3543     \ifcase\count@
3544       \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
3545     \or
3546       \@ifundefined{zth@#1}{\bbl@process@synonym@aux{#1}{0}}{}%
3547     \else
3548       \bbl@process@synonym@aux{#1}{\the\bbl@last}%
3549     \fi}
3550   \ifx\bbl@languages\@undefined % Just a (sensible?) guess
```

```
3551    \chardef\l@english\z@
3552    \chardef\l@USenglish\z@
3553    \chardef\bbl@last\z@
3554    \global\@namedef{bbl@hyphendata@0}{{hyphen.tex}{}}
3555    \gdef\bbl@languages{%
3556       \bbl@elt{english}{0}{hyphen.tex}{}%
3557       \bbl@elt{USenglish}{0}{}{}}
3558  \else
3559    \global\let\bbl@languages@format\bbl@languages
3560    \def\bbl@elt#1#2#3#4{% Remove all except language 0
3561      \ifnum#2>\z@\else
3562        \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
3563      \fi}%
3564    \xdef\bbl@languages{\bbl@languages}%
3565  \fi
3566  \def\bbl@elt#1#2#3#4{\@namedef{zth@#1}{}} % Define flags
3567  \bbl@languages
3568  \openin1=language.dat
3569  \ifeof1
3570    \bbl@warning{I couldn't find language.dat. No additional\\%
3571                 patterns loaded. Reported}%
3572  \else
3573    \loop
3574      \endlinechar\m@ne
3575      \read1 to \bbl@line
3576      \endlinechar`\^^M
3577      \if T\ifeof1F\fi T\relax
3578        \ifx\bbl@line\@empty\else
3579          \edef\bbl@line{\bbl@line\space\space\space}%
3580          \expandafter\bbl@process@line\bbl@line\relax
3581        \fi
3582    \repeat
3583  \fi
3584 \endgroup
3585 \bbl@trace{Macros for reading patterns files}
3586 \def\bbl@get@enc#1:#2:#3\@@@{\def\bbl@hyph@enc{#2}}
3587 \ifx\babelcatcodetablenum\@undefined
3588   \def\babelcatcodetablenum{5211}
3589 \fi
3590 \def\bbl@luapatterns#1#2{%
3591   \bbl@get@enc#1::\@@@
3592   \setbox\z@\hbox\bgroup
3593     \begingroup
3594       \ifx\catcodetable\@undefined
3595         \let\savecatcodetable\luatexsavecatcodetable
3596         \let\initcatcodetable\luatexinitcatcodetable
3597         \let\catcodetable\luatexcatcodetable
3598       \fi
3599       \savecatcodetable\babelcatcodetablenum\relax
3600       \initcatcodetable\numexpr\babelcatcodetablenum+1\relax
3601       \catcodetable\numexpr\babelcatcodetablenum+1\relax
3602       \catcode`\#=6  \catcode`\$=3 \catcode`\&=4 \catcode`\^=7
3603       \catcode`\_=8  \catcode`\{=1 \catcode`\}=2 \catcode`\~=13
3604       \catcode`\@=11 \catcode`\^^I=10 \catcode`\^^J=12
3605       \catcode`\<=12 \catcode`\>=12 \catcode`\*=12 \catcode`\.=12
3606       \catcode`\-=12 \catcode`\/=12 \catcode`\[=12 \catcode`\]=12
3607       \catcode`\`=12 \catcode`\'=12 \catcode`\"=12
3608       \input #1\relax
3609       \catcodetable\babelcatcodetablenum\relax
```

```
3610     \endgroup
3611     \def\bbl@tempa{#2}%
3612     \ifx\bbl@tempa\@empty\else
3613       \input #2\relax
3614     \fi
3615   \egroup}%
3616 \def\bbl@patterns@lua#1{%
3617   \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
3618     \csname l@#1\endcsname
3619     \edef\bbl@tempa{#1}%
3620   \else
3621     \csname l@#1:\f@encoding\endcsname
3622     \edef\bbl@tempa{#1:\f@encoding}%
3623   \fi\relax
3624   \@namedef{lu@texhyphen@loaded@\the\language}{}% Temp
3625   \@ifundefined{bbl@hyphendata@\the\language}%
3626     {\def\bbl@elt##1##2##3##4{%
3627       \ifnum##2=\csname l@\bbl@tempa\endcsname % #2=spanish, dutch:OT1...
3628         \def\bbl@tempb{##3}%
3629         \ifx\bbl@tempb\@empty\else % if not a synonymous
3630           \def\bbl@tempc{{##3}{##4}}%
3631         \fi
3632         \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
3633       \fi}%
3634     \bbl@languages
3635     \@ifundefined{bbl@hyphendata@\the\language}%
3636       {\bbl@info{No hyphenation patterns were set for\\%
3637                 language '\bbl@tempa'. Reported}}%
3638       {\expandafter\expandafter\expandafter\bbl@luapatterns
3639         \csname bbl@hyphendata@\the\language\endcsname}}{}}
3640 \endinput\fi
3641 \begingroup
3642 \catcode`\%=12
3643 \catcode`\'=12
3644 \catcode`\"=12
3645 \catcode`\:=12
3646 \directlua{
3647   Babel = Babel or {}
3648   function Babel.bytes(line)
3649     return line:gsub("(.)",
3650       function (chr) return unicode.utf8.char(string.byte(chr)) end)
3651   end
3652   function Babel.begin_process_input()
3653     if luatexbase and luatexbase.add_to_callback then
3654       luatexbase.add_to_callback('process_input_buffer',
3655                                   Babel.bytes,'Babel.bytes')
3656     else
3657       Babel.callback = callback.find('process_input_buffer')
3658       callback.register('process_input_buffer',Babel.bytes)
3659     end
3660   end
3661   function Babel.end_process_input ()
3662     if luatexbase and luatexbase.remove_from_callback then
3663       luatexbase.remove_from_callback('process_input_buffer','Babel.bytes')
3664     else
3665       callback.register('process_input_buffer',Babel.callback)
3666     end
3667   end
3668   function Babel.addpatterns(pp, lg)
```

```
3669      local lg = lang.new(lg)
3670      local pats = lang.patterns(lg) or ''
3671      lang.clear_patterns(lg)
3672      for p in pp:gmatch('[^%s]+') do
3673        ss = ''
3674        for i in string.utfcharacters(p:gsub('%d', '')) do
3675          ss = ss .. '%d?' .. i
3676        end
3677        ss = ss:gsub('^%%d%?%.', '%%.') .. '%d?'
3678        ss = ss:gsub('%.%%d%?$', '%%.')
3679        pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
3680        if n == 0 then
3681          tex.sprint(
3682            [[\string\csname\space bbl@info\endcsname{New pattern: ]]
3683            .. p .. [[}]])
3684          pats = pats .. ' ' .. p
3685        else
3686          tex.sprint(
3687            [[\string\csname\space bbl@info\endcsname{Renew pattern: ]]
3688            .. p .. [[}]])
3689        end
3690      end
3691      lang.patterns(lg, pats)
3692    end
3693 }
3694 \endgroup
3695 \def\BabelStringsDefault{unicode}
3696 \let\luabbl@stop\relax
3697 \AddBabelHook{luatex}{encodedcommands}{%
3698   \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
3699   \ifx\bbl@tempa\bbl@tempb\else
3700     \directlua{Babel.begin_process_input()}%
3701     \def\luabbl@stop{%
3702       \directlua{Babel.end_process_input()}}%
3703   \fi}%
3704 \AddBabelHook{luatex}{stopcommands}{%
3705   \luabbl@stop
3706   \let\luabbl@stop\relax}
3707 \AddBabelHook{luatex}{patterns}{%
3708   \@ifundefined{bbl@hyphendata@\the\language}%
3709     {\def\bbl@elt##1##2##3##4{%
3710       \ifnum##2=\csname l@##2\endcsname % #2=spanish, dutch:OT1...
3711         \def\bbl@tempb{##3}%
3712         \ifx\bbl@tempb\@empty\else % if not a synonymous
3713           \def\bbl@tempc{{##3}{##4}}%
3714         \fi
3715         \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
3716       \fi}%
3717      \bbl@languages
3718      \@ifundefined{bbl@hyphendata@\the\language}%
3719        {\bbl@info{No hyphenation patterns were set for\\%
3720                  language '#2'. Reported}}%
3721        {\expandafter\expandafter\expandafter\bbl@luapatterns
3722          \csname bbl@hyphendata@\the\language\endcsname}}{}%
3723   \@ifundefined{bbl@patterns@}{}{%
3724     \begingroup
3725       \bbl@xin@{,\number\language,}{,\bbl@pttnlist}%
3726       \ifin@\else
3727         \ifx\bbl@patterns@\@empty\else
```

144

```
3728            \directlua{ Babel.addpatterns(
3729                [[\bbl@patterns@]], \number\language) }%
3730          \fi
3731          \@ifundefined{bbl@patterns@#1}%
3732            \@empty
3733            {\directlua{ Babel.addpatterns(
3734                  [[\space\csname bbl@patterns@#1\endcsname]],
3735                  \number\language) }}%
3736          \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
3737        \fi
3738      \endgroup}}
3739 \AddBabelHook{luatex}{everylanguage}{%
3740   \def\process@language##1##2##3{%
3741     \def\process@line####1####2 ####3 ####4 {}}}
3742 \AddBabelHook{luatex}{loadpatterns}{%
3743   \input #1\relax
3744   \expandafter\gdef\csname bbl@hyphendata@\the\language\endcsname
3745     {{#1}{}}}
3746 \AddBabelHook{luatex}{loadexceptions}{%
3747   \input #1\relax
3748   \def\bbl@tempb##1##2{{##1}{#1}}%
3749   \expandafter\xdef\csname bbl@hyphendata@\the\language\endcsname
3750     {\expandafter\expandafter\expandafter\bbl@tempb
3751      \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.

```
3752 \@onlypreamble\babelpatterns
3753 \AtEndOfPackage{%
3754   \newcommand\babelpatterns[2][\@empty]{%
3755     \ifx\bbl@patterns@\relax
3756       \let\bbl@patterns@\@empty
3757     \fi
3758     \ifx\bbl@pttnlist\@empty\else
3759       \bbl@warning{%
3760         You must not intermingle \string\selectlanguage\space and\\%
3761         \string\babelpatterns\space or some patterns will not\\%
3762         be taken into account. Reported}%
3763     \fi
3764     \ifx\@empty#1%
3765       \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
3766     \else
3767       \edef\bbl@tempb{\zap@space#1 \@empty}%
3768       \bbl@for\bbl@tempa\bbl@tempb{%
3769         \bbl@fixname\bbl@tempa
3770         \bbl@iflanguage\bbl@tempa{%
3771           \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
3772             \@ifundefined{bbl@patterns@\bbl@tempa}%
3773               \@empty
3774               {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
3775             #2}}}%
3776     \fi}}
```

14.4   Southeast Asian scripts

In progress. Replace regular (ie, implicit) discretionaries by spaceskips, based on the
previous glyph (which I think makes sense, because the hyphen and the previous char go

always together). Other discretionaries are not touched.

For the moment, only 3 SA languages are activated by default (see Unicode UAX 14).

```
3777 \def\bbl@intraspace#1 #2 #3\@@{%
3778   \directlua{
3779     Babel = Babel or {}
3780     Babel.intraspaces = Babel.intraspaces or {}
3781     Babel.intraspaces['\csname bbl@sbcp@\languagename\endcsname'] = %
3782         {b = #1, p = #2, m = #3}
3783   }}
3784 \def\bbl@intrapenalty#1\@@{%
3785   \directlua{
3786     Babel = Babel or {}
3787     Babel.intrapenalties = Babel.intrapenalties or {}
3788     Babel.intrapenalties['\csname bbl@sbcp@\languagename\endcsname'] = #1
3789   }}
3790 \begingroup
3791 \catcode`\%=12
3792 \catcode`\^=14
3793 \catcode`\'=12
3794 \catcode`\~=12
3795 \gdef\bbl@seaintraspace{^
3796   \let\bbl@seaintraspace\relax
3797   \directlua{
3798     Babel = Babel or {}
3799     Babel.sea_ranges = Babel.sea_ranges or {}
3800     function Babel.set_chranges (script, chrng)
3801       local c = 0
3802       for s, e in string.gmatch(chrng..' ', '(.-)%.%.(.-)%s') do
3803         Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
3804         c = c + 1
3805       end
3806     end
3807     function Babel.sea_disc_to_space (head)
3808       local sea_ranges = Babel.sea_ranges
3809       local last_char = nil
3810       local quad = 655360        ^^ 10 pt = 655360 = 10 * 65536
3811       for item in node.traverse(head) do
3812         local i = item.id
3813         if i == node.id'glyph' then
3814           last_char = item
3815         elseif i == 7 and item.subtype == 3 and last_char
3816             and last_char.char > 0x0C99 then
3817           quad = font.getfont(last_char.font).size
3818           for lg, rg in pairs(sea_ranges) do
3819             if last_char.char > rg[1] and last_char.char < rg[2] then
3820               lg = lg:sub(1, 4)
3821               local intraspace = Babel.intraspaces[lg]
3822               local intrapenalty = Babel.intrapenalties[lg]
3823               local n
3824               if intrapenalty ~= 0 then
3825                 n = node.new(14, 0)      ^^ penalty
3826                 n.penalty = intrapenalty
3827                 node.insert_before(head, item, n)
3828               end
3829               n = node.new(12, 13)      ^^ (glue, spaceskip)
3830               node.setglue(n, intraspace.b * quad,
3831                               intraspace.p * quad,
3832                               intraspace.m * quad)
```

```
3833            node.insert_before(head, item, n)
3834            node.remove(head, item)
3835          end
3836        end
3837      end
3838    end
3839    end
3840    luatexbase.add_to_callback('hyphenate',
3841      function (head, tail)
3842        lang.hyphenate(head)
3843        Babel.sea_disc_to_space(head)
3844      end,
3845      'Babel.sea_disc_to_space')
3846 }}
3847 \endgroup
```

Common stuff.

```
3848 \AddBabelHook{luatex}{loadkernel}{%
3849 ⟨⟨Restore Unicode catcodes before loading patterns⟩⟩}
3850 \ifx\DisableBabelHook\@undefined\endinput\fi
3851 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
3852 \DisableBabelHook{babel-fontspec}
3853 ⟨⟨Font selection⟩⟩
```

14.5   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.

```
3854 \bbl@trace{Redefinitions for bidi layout}
3855 \ifx\@eqnnum\@undefined\else
3856  \ifx\bbl@attr@dir\@undefined\else
3857    \edef\@eqnnum{{%
3858      \unexpanded{\ifcase\bbl@attr@dir\else\bbl@textdir\@ne\fi}%
3859      \unexpanded\expandafter{\@eqnnum}}}
3860  \fi
3861 \fi
3862 \ifx\bbl@opt@layout\@nnil\endinput\fi  % if no layout
3863 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
3864  \def\bbl@nextfake#1{%  non-local changes - always inside a group!
3865    \bbl@exp{%
3866      \mathdir\the\bodydir
3867      #1%              Once entered in math, set boxes to restore values
3868      \everyvbox{%
3869        \the\everyvbox
3870        \bodydir\the\bodydir
3871        \mathdir\the\mathdir
3872        \everyhbox{\the\everyhbox}%
3873        \everyvbox{\the\everyvbox}}%
3874      \everyhbox{%
3875        \the\everyhbox
```

147

```
3876        \bodydir\the\bodydir
3877        \mathdir\the\mathdir
3878        \everyhbox{\the\everyhbox}%
3879        \everyvbox{\the\everyvbox}}}}%
3880   \def\@hangfrom#1{%
3881     \setbox\@tempboxa\hbox{{#1}}%
3882     \hangindent\wd\@tempboxa
3883     \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
3884       \shapemode\@ne
3885     \fi
3886     \noindent\box\@tempboxa}
3887 \fi
3888 \IfBabelLayout{tabular}
3889   {\def\@tabular{%
3890     \leavevmode\hbox\bgroup\bbl@nextfake$%   %$
3891     \let\@acol\@tabacol       \let\@classz\@tabclassz
3892     \let\@classiv\@tabclassiv \let\\\@tabularcr\@tabarray}}
3893   {}
3894 \IfBabelLayout{lists}
3895   {\def\list#1#2{%
3896     \ifnum \@listdepth >5\relax
3897       \@toodeep
3898     \else
3899       \global\advance\@listdepth\@ne
3900     \fi
3901     \rightmargin\z@
3902     \listparindent\z@
3903     \itemindent\z@
3904     \csname @list\romannumeral\the\@listdepth\endcsname
3905     \def\@itemlabel{#1}%
3906     \let\makelabel\@mklab
3907     \@nmbrlistfalse
3908     #2\relax
3909     \@trivlist
3910     \parskip\parsep
3911     \parindent\listparindent
3912     \advance\linewidth -\rightmargin
3913     \advance\linewidth -\leftmargin
3914     \advance\@totalleftmargin \leftmargin
3915     \parshape \@ne
3916     \@totalleftmargin \linewidth
3917     \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
3918       \shapemode\tw@
3919     \fi
3920     \ignorespaces}}
3921   {}
```

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`.

```
3922 \IfBabelLayout{counters}%
3923   {\def\@textsuperscript#1{{% lua has separate settings for math
3924     \m@th
3925     \mathdir\pagedir % required with basic-r; ok with default, too
3926     \ensuremath{^{\mbox {\fontsize \sf@size \z@ #1}}}}}%
3927   \let\bbl@latinarabic=\@arabic
3928   \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
3929   \@ifpackagewith{babel}{bidi=default}%
3930     {\let\bbl@asciiroman=\@roman
```

```
3931        \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciiroman#1}}}%
3932        \let\bbl@asciiRoman=\@Roman
3933        \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}%
3934        \def\labelenumii{)\theenumii(}%
3935        \def\p@enumiii{\p@enumii)\theenumii(}}{}}{}
3936 ⟨⟨Footnote changes⟩⟩
3937 \IfBabelLayout{footnotes}%
3938   {\BabelFootnote\footnote\languagename{}{}%
3939    \BabelFootnote\localfootnote\languagename{}{}%
3940    \BabelFootnote\mainfootnote{}{}{}}
3941   {}
```

Some LaTeX macros use internally the math mode for text formatting. They have very little in common and are grouped here, as a single option.

```
3942 \IfBabelLayout{extras}%
3943   {\def\underline#1{%
3944      \relax
3945      \ifmmode\@@underline{#1}%
3946      \else\bbl@nextfake$\@@underline{\hbox{#1}}\m@th$\relax\fi}%
3947    \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
3948      \if b\expandafter\@car\f@series\@nil\boldmath\fi
3949      \babelsublr{%
3950        \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}}}
3951   {}
3952 ⟨/luatex⟩
```

14.6 Auto bidi with `basic` **and** `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?)

```
3953 ⟨∗basic-r⟩
3954 Babel = Babel or {}
3955
3956 require('babel-bidi.lua')
3957
3958 local characters = Babel.characters
3959 local ranges = Babel.ranges
3960
3961 local DIR = node.id("dir")
3962
3963 local function dir_mark(head, from, to, outer)
3964   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
3965   local d = node.new(DIR)
3966   d.dir = '+' .. dir
3967   node.insert_before(head, from, d)
3968   d = node.new(DIR)
3969   d.dir = '-' .. dir
3970   node.insert_after(head, to, d)
3971 end
3972
3973 function Babel.pre_otfload_v(head)
3974   -- head = Babel.numbers(head)
3975   head = Babel.bidi(head, true)
3976   return head
3977 end
3978
3979 function Babel.pre_otfload_h(head)
3980   -- head = Babel.numbers(head)
3981   head = Babel.bidi(head, false)
3982   return head
3983 end
3984
3985 function Babel.bidi(head, ispar)
3986   local first_n, last_n        -- first and last char with nums
3987   local last_es                -- an auxiliary 'last' used with nums
3988   local first_d, last_d        -- first and last char in L/R block
3989   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):

```
3990   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
3991   local strong_lr = (strong == 'l') and 'l' or 'r'
3992   local outer = strong
3993
3994   local new_dir = false
3995   local first_dir = false
3996   local inmath = false
3997
3998   local last_lr
3999
4000   local type_n = ''
4001
4002   for item in node.traverse(head) do
4003
4004     -- three cases: glyph, dir, otherwise
4005     if item.id == node.id'glyph'
```

150

```
4006        or (item.id == 7 and item.subtype == 2) then
4007
4008        local itemchar
4009        if item.id == 7 and item.subtype == 2 then
4010          itemchar = item.replace.char
4011        else
4012          itemchar = item.char
4013        end
4014        local chardata = characters[itemchar]
4015        dir = chardata and chardata.d or nil
4016        if not dir then
4017          for nn, et in ipairs(ranges) do
4018            if itemchar < et[1] then
4019              break
4020            elseif itemchar <= et[2] then
4021              dir = et[3]
4022              break
4023            end
4024          end
4025        end
4026        dir = dir or 'l'
4027        if inmath then dir = ('TRT' == tex.mathdir) and 'r' or 'l' end
```

Next is based on the assumption babel sets the language AND switches the script with its dir. We treat a language block as a separate Unicode sequence. The following piece of code is executed at the first glyph after a 'dir' node. We don't know the current language until then.

```
4028        if new_dir then
4029          attr_dir = 0
4030          for at in node.traverse(item.attr) do
4031            if at.number == luatexbase.registernumber'bbl@attr@dir' then
4032              attr_dir = at.value % 3
4033            end
4034          end
4035          if attr_dir == 1 then
4036            strong = 'r'
4037          elseif attr_dir == 2 then
4038            strong = 'al'
4039          else
4040            strong = 'l'
4041          end
4042          strong_lr = (strong == 'l') and 'l' or 'r'
4043          outer = strong_lr
4044          new_dir = false
4045        end
4046
4047        if dir == 'nsm' then dir = strong end              -- W1
```

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

```
4048        dir_real = dir                -- We need dir_real to set strong below
4049        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:

```
4050        if strong == 'al' then
4051          if dir == 'en' then dir = 'an' end              -- W2
4052          if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
4053          strong_lr = 'r'                                  -- W3
4054        end
```

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

```
4055    elseif item.id == node.id'dir' and not inmath then
4056      new_dir = true
4057      dir = nil
4058    elseif item.id == node.id'math' then
4059      inmath = (item.subtype == 0)
4060    else
4061      dir = nil            -- Not a char
4062    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>.

```
4063    if dir == 'en' or dir == 'an' or dir == 'et' then
4064      if dir ~= 'et' then
4065        type_n = dir
4066      end
4067      first_n = first_n or item
4068      last_n = last_es or item
4069      last_es = nil
4070    elseif dir == 'es' and last_n then -- W3+W6
4071      last_es = item
4072    elseif dir == 'cs' then            -- it's right - do nothing
4073    elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
4074      if strong_lr == 'r' and type_n ~= '' then
4075        dir_mark(head, first_n, last_n, 'r')
4076      elseif strong_lr == 'l' and first_d and type_n == 'an' then
4077        dir_mark(head, first_n, last_n, 'r')
4078        dir_mark(head, first_d, last_d, outer)
4079        first_d, last_d = nil, nil
4080      elseif strong_lr == 'l' and type_n ~= '' then
4081        last_d = last_n
4082      end
4083      type_n = ''
4084      first_n, last_n = nil, nil
4085    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:

```
4086    if dir == 'l' or dir == 'r' then
4087      if dir ~= outer then
4088        first_d = first_d or item
4089        last_d = item
4090      elseif first_d and dir ~= strong_lr then
4091        dir_mark(head, first_d, last_d, outer)
4092        first_d, last_d = nil, nil
4093      end
4094    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.

```
4095      if dir and not last_lr and dir ~= 'l' and outer == 'r' then
4096        item.char = characters[item.char] and
4097                      characters[item.char].m or item.char
4098      elseif (dir or new_dir) and last_lr ~= item then
4099        local mir = outer .. strong_lr .. (dir or outer)
4100        if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
4101          for ch in node.traverse(node.next(last_lr)) do
4102            if ch == item then break end
4103            if ch.id == node.id'glyph' then
4104              ch.char = characters[ch.char].m or ch.char
4105            end
4106          end
4107        end
4108      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).

```
4109      if dir == 'l' or dir == 'r' then
4110        last_lr = item
4111        strong = dir_real              -- Don't search back - best save now
4112        strong_lr = (strong == 'l') and 'l' or 'r'
4113      elseif new_dir then
4114        last_lr = nil
4115      end
4116    end
```

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

```
4117    if last_lr and outer == 'r' then
4118      for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
4119        ch.char = characters[ch.char].m or ch.char
4120      end
4121    end
4122    if first_n then
4123      dir_mark(head, first_n, last_n, outer)
4124    end
4125    if first_d then
4126      dir_mark(head, first_d, last_d, outer)
4127    end
```

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

```
4128    return node.prev(head) or head
4129 end
4130 ⟨/basic-r⟩
```

And here the Lua code for bidi=basic:

```
4131 ⟨*basic⟩
4132 Babel = Babel or {}
4133
4134 Babel.fontmap = Babel.fontmap or {}
4135 Babel.fontmap[0] = {}      -- l
4136 Babel.fontmap[1] = {}      -- r
4137 Babel.fontmap[2] = {}      -- al/an
4138
4139 function Babel.pre_otfload_v(head)
```

```lua
4140    -- head = Babel.numbers(head)
4141    head = Babel.bidi(head, true)
4142    return head
4143 end
4144
4145 function Babel.pre_otfload_h(head, gc, sz, pt, dir)
4146    -- head = Babel.numbers(head)
4147    head = Babel.bidi(head, false, dir)
4148    return head
4149 end
4150
4151 require('babel-bidi.lua')
4152
4153 local characters = Babel.characters
4154 local ranges = Babel.ranges
4155
4156 local DIR = node.id('dir')
4157 local GLYPH = node.id('glyph')
4158
4159 local function insert_implicit(head, state, outer)
4160    local new_state = state
4161    if state.sim and state.eim and state.sim ~= state.eim then
4162      dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
4163      local d = node.new(DIR)
4164      d.dir = '+' .. dir
4165      node.insert_before(head, state.sim, d)
4166      local d = node.new(DIR)
4167      d.dir = '-' .. dir
4168      node.insert_after(head, state.eim, d)
4169    end
4170    new_state.sim, new_state.eim = nil, nil
4171    return head, new_state
4172 end
4173
4174 local function insert_numeric(head, state)
4175    local new
4176    local new_state = state
4177    if state.san and state.ean and state.san ~= state.ean then
4178      local d = node.new(DIR)
4179      d.dir = '+TLT'
4180      _, new = node.insert_before(head, state.san, d)
4181      if state.san == state.sim then state.sim = new end
4182      local d = node.new(DIR)
4183      d.dir = '-TLT'
4184      _, new = node.insert_after(head, state.ean, d)
4185      if state.ean == state.eim then state.eim = new end
4186    end
4187    new_state.san, new_state.ean = nil, nil
4188    return head, new_state
4189 end
4190
4191 -- TODO - \hbox with an explicit dir can lead to wrong results
4192 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
4193 -- was s made to improve the situation, but the problem is the 3-dir
4194 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
4195 -- well.
4196
4197 function Babel.bidi(head, ispar, hdir)
4198    local d    -- d is used mainly for computations in a loop
```

```
4199   local prev_d = ''
4200   local new_d = false
4201
4202   local nodes = {}
4203   local outer_first = nil
4204   local inmath = false
4205
4206   local glue_d = nil
4207   local glue_i = nil
4208
4209   local has_en = false
4210   local first_et = nil
4211
4212   local ATDIR = luatexbase.registernumber'bbl@attr@dir'
4213
4214   local save_outer
4215   local temp = node.get_attribute(head, ATDIR)
4216   if temp then
4217     temp = temp % 3
4218     save_outer = (temp == 0 and 'l') or
4219                  (temp == 1 and 'r') or
4220                  (temp == 2 and 'al')
4221   elseif ispar then              -- Or error? Shouldn't happen
4222     save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
4223   else                           -- Or error? Shouldn't happen
4224     save_outer = ('TRT' == hdir) and 'r' or 'l'
4225   end
4226     -- when the callback is called, we are just _after_ the box,
4227     -- and the textdir is that of the surrounding text
4228   -- if not ispar and hdir ~= tex.textdir then
4229   --   save_outer = ('TRT' == hdir) and 'r' or 'l'
4230   -- end
4231   local outer = save_outer
4232   local last = outer
4233   -- 'al' is only taken into account in the first, current loop
4234   if save_outer == 'al' then save_outer = 'r' end
4235
4236   local fontmap = Babel.fontmap
4237
4238   for item in node.traverse(head) do
4239
4240     -- In what follows, #node is the last (previous) node, because the
4241     -- current one is not added until we start processing the neutrals.
4242
4243     -- three cases: glyph, dir, otherwise
4244     if item.id == GLYPH
4245        or (item.id == 7 and item.subtype == 2) then
4246
4247       local d_font = nil
4248       local item_r
4249       if item.id == 7 and item.subtype == 2 then
4250         item_r = item.replace    -- automatic discs have just 1 glyph
4251       else
4252         item_r = item
4253       end
4254       local chardata = characters[item_r.char]
4255       d = chardata and chardata.d or nil
4256       if not d or d == 'nsm' then
4257         for nn, et in ipairs(ranges) do
```

155

```
4258           if item_r.char < et[1] then
4259             break
4260           elseif item_r.char <= et[2] then
4261             if not d then d = et[3]
4262             elseif d == 'nsm' then d_font = et[3]
4263             end
4264             break
4265           end
4266         end
4267       end
4268       d = d or 'l'
4269       if inmath then d = ('TRT' == tex.mathdir) and 'r' or 'l' end
4270       d_font = d_font or d
4271
4272       d_font = (d_font == 'l' and 0) or
4273                (d_font == 'nsm' and 0) or
4274                (d_font == 'r' and 1) or
4275                (d_font == 'al' and 2) or
4276                (d_font == 'an' and 2) or nil
4277       if d_font and fontmap and fontmap[d_font][item_r.font] then
4278         item_r.font = fontmap[d_font][item_r.font]
4279       end
4280
4281       if new_d then
4282         table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
4283         attr_d = node.get_attribute(item, ATDIR)
4284         attr_d = attr_d % 3
4285         if attr_d == 1 then
4286           outer_first = 'r'
4287           last = 'r'
4288         elseif attr_d == 2 then
4289           outer_first = 'r'
4290           last = 'al'
4291         else
4292           outer_first = 'l'
4293           last = 'l'
4294         end
4295         outer = last
4296         has_en = false
4297         first_et = nil
4298         new_d = false
4299       end
4300
4301       if glue_d then
4302         if (d == 'l' and 'l' or 'r') ~= glue_d then
4303           table.insert(nodes, {glue_i, 'on', nil})
4304         end
4305         glue_d = nil
4306         glue_i = nil
4307       end
4308
4309     elseif item.id == DIR then
4310       d = nil
4311       new_d = true
4312
4313     elseif item.id == node.id'glue' and item.subtype == 13 then
4314       glue_d = d
4315       glue_i = item
4316       d = nil
```

```
4317
4318    elseif item.id == node.id'math' then
4319      inmath = (item.subtype == 0)
4320
4321    else
4322      d = nil
4323    end
4324
4325    -- AL <= EN/ET/ES     -- W2 + W3 + W6
4326    if last == 'al' and d == 'en' then
4327      d = 'an'           -- W3
4328    elseif last == 'al' and (d == 'et' or d == 'es') then
4329      d = 'on'           -- W6
4330    end
4331
4332    -- EN + CS/ES + EN       -- W4
4333    if d == 'en' and #nodes >= 2 then
4334      if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
4335         and nodes[#nodes-1][2] == 'en' then
4336        nodes[#nodes][2] = 'en'
4337      end
4338    end
4339
4340    -- AN + CS + AN        -- W4 too, because uax9 mixes both cases
4341    if d == 'an' and #nodes >= 2 then
4342      if (nodes[#nodes][2] == 'cs')
4343         and nodes[#nodes-1][2] == 'an' then
4344        nodes[#nodes][2] = 'an'
4345      end
4346    end
4347
4348    -- ET/EN               -- W5 + W7->l / W6->on
4349    if d == 'et' then
4350      first_et = first_et or (#nodes + 1)
4351    elseif d == 'en' then
4352      has_en = true
4353      first_et = first_et or (#nodes + 1)
4354    elseif first_et then       -- d may be nil here !
4355      if has_en then
4356        if last == 'l' then
4357          temp = 'l'     -- W7
4358        else
4359          temp = 'en'    -- W5
4360        end
4361      else
4362        temp = 'on'      -- W6
4363      end
4364      for e = first_et, #nodes do
4365        if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
4366      end
4367      first_et = nil
4368      has_en = false
4369    end
4370
4371    if d then
4372      if d == 'al' then
4373        d = 'r'
4374        last = 'al'
4375      elseif d == 'l' or d == 'r' then
```

```
4376          last = d
4377        end
4378        prev_d = d
4379        table.insert(nodes, {item, d, outer_first})
4380      end
4381
4382    outer_first = nil
4383
4384  end
4385
4386  -- TODO -- repeated here in case EN/ET is the last node. Find a
4387  -- better way of doing things:
4388  if first_et then        -- dir may be nil here !
4389    if has_en then
4390      if last == 'l' then
4391        temp = 'l'     -- W7
4392      else
4393        temp = 'en'    -- W5
4394      end
4395    else
4396      temp = 'on'      -- W6
4397    end
4398    for e = first_et, #nodes do
4399      if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
4400    end
4401  end
4402
4403  -- dummy node, to close things
4404  table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
4405
4406  --------------  NEUTRAL  -----------------
4407
4408  outer = save_outer
4409  last = outer
4410
4411  local first_on = nil
4412
4413  for q = 1, #nodes do
4414    local item
4415
4416    local outer_first = nodes[q][3]
4417    outer = outer_first or outer
4418    last = outer_first or last
4419
4420    local d = nodes[q][2]
4421    if d == 'an' or d == 'en' then d = 'r' end
4422    if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
4423
4424    if d == 'on' then
4425      first_on = first_on or q
4426    elseif first_on then
4427      if last == d then
4428        temp = d
4429      else
4430        temp = outer
4431      end
4432      for r = first_on, q - 1 do
4433        nodes[r][2] = temp
4434        item = nodes[r][1]    -- MIRRORING
```

```
4435            if item.id == GLYPH and temp == 'r' then
4436              item.char = characters[item.char].m or item.char
4437            end
4438          end
4439        first_on = nil
4440      end
4441
4442    if d == 'r' or d == 'l' then last = d end
4443  end
4444
4445  -------------  IMPLICIT, REORDER ----------------
4446
4447  outer = save_outer
4448  last = outer
4449
4450  local state = {}
4451  state.has_r = false
4452
4453  for q = 1, #nodes do
4454
4455    local item = nodes[q][1]
4456
4457    outer = nodes[q][3] or outer
4458
4459    local d = nodes[q][2]
4460
4461    if d == 'nsm' then d = last end              -- W1
4462    if d == 'en' then d = 'an' end
4463    local isdir = (d == 'r' or d == 'l')
4464
4465    if outer == 'l' and d == 'an' then
4466      state.san = state.san or item
4467      state.ean = item
4468    elseif state.san then
4469      head, state = insert_numeric(head, state)
4470    end
4471
4472    if outer == 'l' then
4473      if d == 'an' or d == 'r' then      -- im -> implicit
4474        if d == 'r' then state.has_r = true end
4475        state.sim = state.sim or item
4476        state.eim = item
4477      elseif d == 'l' and state.sim and state.has_r then
4478        head, state = insert_implicit(head, state, outer)
4479      elseif d == 'l' then
4480        state.sim, state.eim, state.has_r = nil, nil, false
4481      end
4482    else
4483      if d == 'an' or d == 'l' then
4484        if nodes[q][3] then -- nil except after an explicit dir
4485          state.sim = item  -- so we move sim 'inside' the group
4486        else
4487          state.sim = state.sim or item
4488        end
4489        state.eim = item
4490      elseif d == 'r' and state.sim then
4491        head, state = insert_implicit(head, state, outer)
4492      elseif d == 'r' then
4493        state.sim, state.eim = nil, nil
```

159

```
4494        end
4495      end
4496
4497    if isdir then
4498      last = d                -- Don't search back - best save now
4499    elseif d == 'on' and state.san  then
4500      state.san = state.san or item
4501      state.ean = item
4502    end
4503
4504  end
4505
4506  return node.prev(head) or head
4507 end
4508 ⟨/basic⟩
```

15   The 'nil' language

This 'language' does nothing, except setting the hyphenation patterns to nohyphenation.
For this language currently no special definitions are needed or available.
The macro \LdfInit takes care of preventing that this file is loaded more than once,
checking the category code of the @ sign, etc.

```
4509 ⟨*nil⟩
4510 \ProvidesLanguage{nil}[⟨⟨date⟩⟩ ⟨⟨version⟩⟩ Nil language]
4511 \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.

```
4512 \ifx\l@nohyphenation\@undefined
4513    \@nopatterns{nil}
4514    \adddialect\l@nil0
4515 \else
4516    \let\l@nil\l@nohyphenation
4517 \fi
```

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

```
4518 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}
```

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

```
\captionnil
   \datenil  4519 \let\captionsnil\@empty
4520 \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.

```
4521 \ldf@finish{nil}
4522 ⟨/nil⟩
```

16   Support for Plain TeX (plain.def)

16.1   Not renaming hyphen.tex

As Don Knuth has declared that the filename hyphen.tex may only be used to designate
his version of the american English hyphenation patterns, a new solution has to be found

in order to be able to load hyphenation patterns for other languages in a plain-based TeX-format. When asked he responded:

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

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

The files `bplain.tex` and `blplain.tex` can be used as replacement wrappers around `plain.tex` and `lplain.tex` to 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`

```
4523 ⟨∗bplain | blplain⟩
4524 \catcode`\{=1 % left brace is begin-group character
4525 \catcode`\}=2 % right brace is end-group character
4526 \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…

```
4527 \openin 0 hyphen.cfg
```

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

```
4528 \ifeof0
4529 \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).

```
4530   \let\a\input
```

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

```
4531   \def\input #1 {%
4532     \let\input\a
4533     \a hyphen.cfg
```

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

```
4534     \let\a\undefined
4535   }
4536 \fi
4537 ⟨/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`.

```
4538 ⟨bplain⟩\a plain.tex
4539 ⟨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.

```
4540 ⟨bplain⟩\def\fmtname{babel-plain}
4541 ⟨blplain⟩\def\fmtname{babel-lplain}
```

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

16.2 Emulating some LaTeX features

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

```
4542 ⟨∗plain⟩
4543 \def\@empty{}
4544 \def\loadlocalcfg#1{%
4545   \openin0#1.cfg
4546   \ifeof0
4547     \closein0
4548   \else
4549     \closein0
4550     {\immediate\write16{***********************************}%
4551      \immediate\write16{* Local config file #1.cfg used}%
4552      \immediate\write16{*}%
4553      }
4554     \input #1.cfg\relax
4555   \fi
4556   \@endofldf}
```

16.3 General tools

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

```
4557 \long\def\@firstofone#1{#1}
4558 \long\def\@firstoftwo#1#2{#1}
4559 \long\def\@secondoftwo#1#2{#2}
4560 \def\@nnil{\@nil}
4561 \def\@gobbletwo#1#2{}
4562 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
4563 \def\@star@or@long#1{%
4564   \@ifstar
4565   {\let\l@ngrel@x\relax#1}%
4566   {\let\l@ngrel@x\long#1}}
4567 \let\l@ngrel@x\relax
4568 \def\@car#1#2\@nil{#1}
4569 \def\@cdr#1#2\@nil{#2}
4570 \let\@typeset@protect\relax
4571 \let\protected@edef\edef
4572 \long\def\@gobble#1{}
4573 \edef\@backslashchar{\expandafter\@gobble\string\\}
4574 \def\strip@prefix#1>{}
4575 \def\g@addto@macro#1#2{{%
4576   \toks@\expandafter{#1#2}%
4577   \xdef#1{\the\toks@}}}
4578 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
4579 \def\@nameuse#1{\csname #1\endcsname}
4580 \def\@ifundefined#1{%
4581   \expandafter\ifx\csname#1\endcsname\relax
4582     \expandafter\@firstoftwo
4583   \else
4584     \expandafter\@secondoftwo
4585   \fi}
4586 \def\@expandtwoargs#1#2#3{%
4587   \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
4588 \def\zap@space#1 #2{%
4589   #1%
4590   \ifx#2\@empty\else\expandafter\zap@space\fi
4591   #2}
```

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

```
4592 \ifx\@preamblecmds\@undefined
4593   \def\@preamblecmds{}
4594 \fi
4595 \def\@onlypreamble#1{%
4596   \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
4597     \@preamblecmds\do#1}}
4598 \@onlypreamble\@onlypreamble
```

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

```
4599 \def\begindocument{%
4600   \@begindocumenthook
4601   \global\let\@begindocumenthook\@undefined
4602   \def\do##1{\global\let##1\@undefined}%
4603   \@preamblecmds
4604   \global\let\do\noexpand}

4605 \ifx\@begindocumenthook\@undefined
4606   \def\@begindocumenthook{}
4607 \fi
4608 \@onlypreamble\@begindocumenthook
4609 \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.

```
4610 \def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
4611 \@onlypreamble\AtEndOfPackage
4612 \def\@endofldf{}
4613 \@onlypreamble\@endofldf
4614 \let\bbl@afterlang\@empty
4615 \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.

```
4616 \ifx\if@filesw\@undefined
4617   \expandafter\let\csname if@filesw\expandafter\endcsname
4618     \csname iffalse\endcsname
4619 \fi
```

Mimick LaTeX's commands to define control sequences.

```
4620 \def\newcommand{\@star@or@long\new@command}
4621 \def\new@command#1{%
4622   \@testopt{\@newcommand#1}0}
4623 \def\@newcommand#1[#2]{%
4624   \@ifnextchar [{\@xargdef#1[#2]}%
4625               {\@argdef#1[#2]}}
4626 \long\def\@argdef#1[#2]#3{%
4627   \@yargdef#1\@ne{#2}{#3}}
4628 \long\def\@xargdef#1[#2][#3]#4{%
4629   \expandafter\def\expandafter#1\expandafter{%
4630     \expandafter\@protected@testopt\expandafter #1%
4631     \csname\string#1\expandafter\endcsname{#3}}%
4632   \expandafter\@yargdef \csname\string#1\endcsname
4633   \tw@{#2}{#4}}
4634 \long\def\@yargdef#1#2#3{%
4635   \@tempcnta#3\relax
4636   \advance \@tempcnta \@ne
```

```
4637    \let\@hash@\relax
4638    \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
4639    \@tempcntb #2%
4640    \@whilenum\@tempcntb <\@tempcnta
4641    \do{%
4642      \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
4643      \advance\@tempcntb \@ne}%
4644    \let\@hash@##%
4645    \l@ngrel@x\expandafter\def\expandafter#1\reserved@a}
4646 \def\providecommand{\@star@or@long\provide@command}
4647 \def\provide@command#1{%
4648    \begingroup
4649      \escapechar\m@ne\xdef\@gtempa{{\string#1}}%
4650    \endgroup
4651    \expandafter\@ifundefined\@gtempa
4652      {\def\reserved@a{\new@command#1}}%
4653      {\let\reserved@a\relax
4654       \def\reserved@a{\new@command\reserved@a}}%
4655    \reserved@a}%

4656 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
4657 \def\declare@robustcommand#1{%
4658    \edef\reserved@a{\string#1}%
4659    \def\reserved@b{#1}%
4660    \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
4661    \edef#1{%
4662      \ifx\reserved@a\reserved@b
4663        \noexpand\x@protect
4664        \noexpand#1%
4665      \fi
4666      \noexpand\protect
4667      \expandafter\noexpand\csname
4668        \expandafter\@gobble\string#1 \endcsname
4669    }%
4670    \expandafter\new@command\csname
4671      \expandafter\@gobble\string#1 \endcsname
4672 }
4673 \def\x@protect#1{%
4674    \ifx\protect\@typeset@protect\else
4675      \@x@protect#1%
4676    \fi
4677 }
4678 \def\@x@protect#1\fi#2#3{%
4679    \fi\protect#1%
4680 }
```

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`.

```
4681 \def\bbl@tempa{\csname newif\endcsname\ifin@}
4682 \ifx\in@\@undefined
4683   \def\in@#1#2{%
4684     \def\in@@##1#1##2##3\in@@{%
4685       \ifx\in@##2\in@false\else\in@true\fi}%
4686     \in@@#2#1\in@\in@@}
4687 \else
4688   \let\bbl@tempa\@empty
4689 \fi
```

```
4690 \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).

```
4691 \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.

```
4692 \def\@ifl@aded#1#2#3#4{}
```

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

```
4693 \ifx\@tempcnta\@undefined
4694   \csname newcount\endcsname\@tempcnta\relax
4695 \fi
4696 \ifx\@tempcntb\@undefined
4697   \csname newcount\endcsname\@tempcntb\relax
4698 \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).

```
4699 \ifx\bye\@undefined
4700   \advance\count10 by -2\relax
4701 \fi
4702 \ifx\@ifnextchar\@undefined
4703   \def\@ifnextchar#1#2#3{%
4704     \let\reserved@d=#1%
4705     \def\reserved@a{#2}\def\reserved@b{#3}%
4706     \futurelet\@let@token\@ifnch}
4707   \def\@ifnch{%
4708     \ifx\@let@token\@sptoken
4709       \let\reserved@c\@xifnch
4710     \else
4711       \ifx\@let@token\reserved@d
4712         \let\reserved@c\reserved@a
4713       \else
4714         \let\reserved@c\reserved@b
4715       \fi
4716     \fi
4717     \reserved@c}
4718   \def\:{\let\@sptoken= } \:  % this makes \@sptoken a space token
4719   \def\:{\@xifnch} \expandafter\def\: {\futurelet\@let@token\@ifnch}
4720 \fi
4721 \def\@testopt#1#2{%
4722   \@ifnextchar[{#1}{#1[#2]}}
4723 \def\@protected@testopt#1{%
4724   \ifx\protect\@typeset@protect
4725     \expandafter\@testopt
4726   \else
4727     \@x@protect#1%
4728   \fi}
4729 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
4730     #2\relax}\fi}
```

```
4731 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
4732         \else\expandafter\@gobble\fi{#1}}
```

16.4 Encoding related macros

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

```
4733 \def\DeclareTextCommand{%
4734    \@dec@text@cmd\providecommand
4735 }
4736 \def\ProvideTextCommand{%
4737    \@dec@text@cmd\providecommand
4738 }
4739 \def\DeclareTextSymbol#1#2#3{%
4740    \@dec@text@cmd\chardef#1{#2}#3\relax
4741 }
4742 \def\@dec@text@cmd#1#2#3{%
4743    \expandafter\def\expandafter#2%
4744       \expandafter{%
4745          \csname#3-cmd\expandafter\endcsname
4746          \expandafter#2%
4747          \csname#3\string#2\endcsname
4748       }%
4749 %    \let\@ifdefinable\@rc@ifdefinable
4750    \expandafter#1\csname#3\string#2\endcsname
4751 }
4752 \def\@current@cmd#1{%
4753   \ifx\protect\@typeset@protect\else
4754      \noexpand#1\expandafter\@gobble
4755   \fi
4756 }
4757 \def\@changed@cmd#1#2{%
4758    \ifx\protect\@typeset@protect
4759       \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
4760          \expandafter\ifx\csname ?\string#1\endcsname\relax
4761             \expandafter\def\csname ?\string#1\endcsname{%
4762                \@changed@x@err{#1}%
4763             }%
4764          \fi
4765          \global\expandafter\let
4766            \csname\cf@encoding \string#1\expandafter\endcsname
4767            \csname ?\string#1\endcsname
4768       \fi
4769       \csname\cf@encoding\string#1%
4770          \expandafter\endcsname
4771    \else
4772       \noexpand#1%
4773    \fi
4774 }
4775 \def\@changed@x@err#1{%
4776    \errhelp{Your command will be ignored, type <return> to proceed}%
4777    \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
4778 \def\DeclareTextCommandDefault#1{%
4779    \DeclareTextCommand#1?%
4780 }
4781 \def\ProvideTextCommandDefault#1{%
4782    \ProvideTextCommand#1?%
4783 }
4784 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
```

```
4785 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
4786 \def\DeclareTextAccent#1#2#3{%
4787   \DeclareTextCommand#1{#2}[1]{\accent#3 ##1}
4788 }
4789 \def\DeclareTextCompositeCommand#1#2#3#4{%
4790     \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
4791     \edef\reserved@b{\string##1}%
4792     \edef\reserved@c{%
4793       \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
4794     \ifx\reserved@b\reserved@c
4795       \expandafter\expandafter\expandafter\ifx
4796         \expandafter\@car\reserved@a\relax\relax\@nil
4797         \@text@composite
4798       \else
4799         \edef\reserved@b##1{%
4800           \def\expandafter\noexpand
4801             \csname#2\string#1\endcsname####1{%
4802             \noexpand\@text@composite
4803               \expandafter\noexpand\csname#2\string#1\endcsname
4804               ####1\noexpand\@empty\noexpand\@text@composite
4805               {##1}%
4806           }%
4807         }%
4808         \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
4809       \fi
4810       \expandafter\def\csname\expandafter\string\csname
4811         #2\endcsname\string#1-\string#3\endcsname{#4}
4812   \else
4813     \errhelp{Your command will be ignored, type <return> to proceed}%
4814     \errmessage{\string\DeclareTextCompositeCommand\space used on
4815         inappropriate command \protect#1}
4816   \fi
4817 }
4818 \def\@text@composite#1#2#3\@text@composite{%
4819   \expandafter\@text@composite@x
4820     \csname\string#1-\string#2\endcsname
4821 }
4822 \def\@text@composite@x#1#2{%
4823   \ifx#1\relax
4824     #2%
4825   \else
4826     #1%
4827   \fi
4828 }
4829 %
4830 \def\@strip@args#1:#2-#3\@strip@args{#2}
4831 \def\DeclareTextComposite#1#2#3#4{%
4832   \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
4833   \bgroup
4834     \lccode`\@=#4%
4835     \lowercase{%
4836   \egroup
4837     \reserved@a @%
4838   }%
4839 }
4840 %
4841 \def\UseTextSymbol#1#2{%
4842 %    \let\@curr@enc\cf@encoding
4843 %    \@use@text@encoding{#1}%
```

167

```
4844    #2%
4845 %   \@use@text@encoding\@curr@enc
4846 }
4847 \def\UseTextAccent#1#2#3{%
4848 %   \let\@curr@enc\cf@encoding
4849 %   \@use@text@encoding{#1}%
4850 %   #2{\@use@text@encoding\@curr@enc\selectfont#3}%
4851 %   \@use@text@encoding\@curr@enc
4852 }
4853 \def\@use@text@encoding#1{%
4854 %   \edef\f@encoding{#1}%
4855 %   \xdef\font@name{%
4856 %       \csname\curr@fontshape/\f@size\endcsname
4857 %   }%
4858 %   \pickup@font
4859 %   \font@name
4860 %   \@@enc@update
4861 }
4862 \def\DeclareTextSymbolDefault#1#2{%
4863    \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
4864 }
4865 \def\DeclareTextAccentDefault#1#2{%
4866    \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
4867 }
4868 \def\cf@encoding{OT1}
```

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

```
4869 \DeclareTextAccent{\"}{OT1}{127}
4870 \DeclareTextAccent{\'}{OT1}{19}
4871 \DeclareTextAccent{\^}{OT1}{94}
4872 \DeclareTextAccent{\`}{OT1}{18}
4873 \DeclareTextAccent{\~}{OT1}{126}
```

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

```
4874 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
4875 \DeclareTextSymbol{\textquotedblright}{OT1}{`\"}
4876 \DeclareTextSymbol{\textquoteleft}{OT1}{`\`}
4877 \DeclareTextSymbol{\textquoteright}{OT1}{`\'}
4878 \DeclareTextSymbol{\i}{OT1}{16}
4879 \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.

```
4880 \ifx\scriptsize\@undefined
4881    \let\scriptsize\sevenrm
4882 \fi
4883 ⟨/plain⟩
```

17 Acknowledgements

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

References

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

[2] 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.