Babel

Localization and internationalization

Unicode
T_EX
pdfT_EX
LuaT_EX
XeT_EX

Contents

Troubleshoooting

Part I

User guide

- This user guide focuses on internationalization and localization with 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 , and there are some notes for the latest versions in the babel wiki. The most recent features could be still unstable. Please, report any issues you find in GitHub, 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 mail list. You can follow the development of babel in GitHub (which provides many sample files, too). If you are the author of a package, feel free to send to me a few test files which I'll add to mine, so that possible issues could be caught in the development phase.

- 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 does *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 standard mechanism for this purpose, namely, passing that language as an optional argument. In addition, you may want to set the font and input encodings. Many languages are compatible with xetex and luatex. With them you can use babel to localize the documents. When these engines are used, the Latin script is covered by default in current LaTeX (provided the document encoding is UTF-8), because the font loader is preloaded and the font is switched to `lmroman`. Other scripts require loading fontspec. You may want to set the font attributes with fontspec, too.

EXAMPLE Here is a simple full example for "traditional" TeX engines (see below for xetex and luatex). The packages `fontenc` and `inputenc` do not belong to babel, but they are included in the example because typically you will need them (however, the package inputenc may be omitted with LaTeX ≥ 2018-04-01 if the encoding is UTF-8):

```
PDFTEX
    \documentclass{article}

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

    \usepackage[french]{babel}

    \begin{document}

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

    \end{document}
```

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

LUATEX/XETEX

```
\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}
```

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

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

Or the more explanatory:

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

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

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 a list of the required languages as package or class options. The last language is considered the main one, activated by default. Sometimes, the main language changes the document layout (eg, spanish and french).

EXAMPLE In LaTeX, the preamble of the document:

```
\documentclass{article}
\usepackage[dutch,english]{babel}
```

would tell LaTeX that the document would be written in two languages, Dutch and English, and that English would be the first language in use, and the main one.

You can also set the main language explicitly, but it is discouraged except if there a real reason to do so:

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

Examples of cases where main is useful are the following.

NOTE Some classes load babel with a hardcoded language option. Sometimes, the main language 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.

PDFTEX

```
\documentclass{article}

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

\usepackage[english,french]{babel}

\begin{document}

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

\selectlanguage{english}

And an English paragraph, with a short text in
\foreignlanguage{french}{français}.

\end{document}
```

EXAMPLE With xetex and luatex, the following bilingual, single script document in UTF-8 encoding just prints a couple of 'captions' and \today in Danish and Vietnamese. No additional packages are required.

LUATEX/XETEX

```
\documentclass{article}

\usepackage[vietnamese,danish]{babel}

\begin{document}

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

\selectlanguage{vietnamese}

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

\end{document}
```

1.3 Mostly monolingual documents

New 3.39 Very often, multilingual documents consist of a main language with small pieces of text in another languages (words, idioms, short sentences). Typically, all you need is to set the line breaking rules and, perhaps, the font. In such a case, babel now does not require declaring these secondary languages explicitly, because the basic settings are loaded on the fly when the language is selected (and also when provided in the optional argument of \babelfont, if used.)
This is particularly useful, too, when there are short texts of this kind coming from an external source whose contents are not known on beforehand (for example, titles in a bibliography). At this regard, it is worth remembering that \babelfont does not load any font until required, so that it can be used just in case.

EXAMPLE A trivial document is:

```
\documentclass{article}
\usepackage[english]{babel}

\babelfont[russian]{rm}{FreeSerif}

\begin{document}

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

\end{document}
```

1.4 Modifiers

New 3.9c The basic behavior of some languages can be modified when loading babel by means of *modifiers*. They are set after the language name, and are prefixed with a dot (only when the language is set as package option – neither global options nor the main key accepts them). An example is (spaces are not significant and they can be added or removed):[1]

```
\usepackage[latin.medieval, spanish.notilde.lcroman, danish]{babel}
```

Attributes (described below) are considered modifiers, ie, you can set an attribute by including it in the list of modifiers. However, modifiers are a more general mechanism.

1.5 Troubleshooting

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

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

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

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

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

---

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

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

[3]In old versions the error read "You haven't loaded the language LANG yet".

1.6 Plain

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

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

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

1.7 Basic language selectors

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

`\selectlanguage`   {⟨*language*⟩}

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

```
\selectlanguage{german}
```

This command can be used as environment, too.

NOTE For "historical reasons", a macro name is converted to a language name without the leading `\`; in other words, `\selectlanguage{\german}` is equivalent to `\selectlanguage{german}`. Using a macro instead of a "real" name is deprecated.

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

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

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

`\foreignlanguage`   {⟨*language*⟩}{⟨*text*⟩}

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

---

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

1.8   Auxiliary language selectors

`\begin{otherlanguage}` {⟨*language*⟩} … `\end{otherlanguage}`

The environment `otherlanguage` does basically the same as `\selectlanguage`, except that language change is (mostly) local to the environment.

Actually, there might be some non-local changes, as this environment is roughly equivalent to:

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

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

Spaces after the environment are ignored.

`\begin{otherlanguage*}` {⟨*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.

---

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

1.10 Shorthands

A *shorthand* is a sequence of one or two characters that expands to arbitrary TeX code. Shorthands can be used for different kinds of things, for example: (1) in some languages shorthands such as "a are defined to be able to hyphenate the word if the encoding is OT1; (2) in some languages shorthands such as ! are used to insert the right amount of white space; (3) several kinds of discretionaries and breaks can be inserted easily with "-, "=, etc. The package inputenc as well as xetex and luatex have alleviated entering non-ASCII characters, but minority languages and some kinds of text can still require characters not directly available on the keyboards (and sometimes not even as separated or precomposed Unicode characters). As to the point 2, now pdfTeX provides \knbccode, and luatex can manipulate the glyph list. Tools for point 3 can be still very useful in general.
There are three levels of shorthands: *user*, *language*, and *system* (by order of precedence). Version 3.9 introduces the *language user* level on top of the user level, as described below. In most cases, you will use only shorthands provided by languages.

NOTE Note the following:

1. Activated chars used for two-char shorthands cannot be followed by a closing brace } and the spaces following are gobbled. With one-char shorthands (eg, :), they are preserved.

2. If on a certain level (system, language, user) there is a one-char shorthand, two-char ones starting with that char and on the same level are ignored.

3. Since they are active, a shorthand cannot contain the same character in its definition (except if it is deactivated with, eg, \string).

TROUBLESHOOTING A typical error when using shorthands is the following:

```
! Argument of \language@active@arg" has an extra }.
```

It means there is a closing brace just after a shorthand, which is not allowed (eg, "}). Just add {} after (eg, "{}}).

\shorthandon {⟨*shorthands-list*⟩}
\shorthandoff *****{⟨*shorthands-list*⟩}

It is sometimes necessary to switch a shorthand character off temporarily, because it must be used in an entirely different way. For this purpose, the user commands \shorthandoff and \shorthandon are provided. They each take a list of characters as their arguments. The command \shorthandoff sets the \catcode for each of the characters in its argument to other (12); the command \shorthandon sets the \catcode to active (13). Both commands only work on 'known' shorthand characters.
New 3.9a However, \shorthandoff does not behave as you would expect with characters like ~ or ^, because they usually are not "other". For them \shorthandoff* is provided, so that with

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

~ is still active, very likely with the meaning of a non-breaking space, and ^ is the superscript character. The catcodes used are those when the shorthands are defined, usually when language files are loaded.
If you do not need shorthands, or prefer an alternative approach of your own, you may want to switch them off with the package option shorthands=off, as described below.

\useshorthands **`*{⟨char⟩}`**

The command `\useshorthands` initiates the definition of user-defined shorthand sequences. It has one argument, the character that starts these personal shorthands. New 3.9a User shorthands are not always alive, as they may be deactivated by languages (for example, if you use " for your user shorthands and switch from german to french, they stop working). Therefore, a starred version `\useshorthands*{⟨char⟩}` is provided, which makes sure shorthands are always activated.

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

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

The command `\defineshorthand` takes two arguments: the first is a one- or two-character shorthand sequence, and the second is the code the shorthand should expand to. New 3.9a An optional argument allows to (re)define language and system shorthands (some languages do not activate shorthands, so you may want to add `\languageshorthands{⟨lang⟩}` to the corresponding `\extras⟨lang⟩`, as explained below). By default, user shorthands are (re)defined.

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

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

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

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

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

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

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

\languageshorthands **`{⟨language⟩}`**

The command `\languageshorthands` can be used to switch the shorthands on the language level. It takes one argument, the name of a language or `none` (the latter does what its name suggests).[6] Note that for this to work the language should have been specified as an option when loading the babel package. For example, you can use in english the shorthands defined by ngerman with

---

[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.

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

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

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

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

\babelshorthand {⟨*shorthand*⟩}

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

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

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

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

---

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

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

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

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

The command `\aliasshorthand` can be used to let another character perform the same functions as the default shorthand character. If one prefers for example to use the character / over " in typing Polish texts, this can be achieved by entering `\aliasshorthand{"}{/}`. For the reasons in the warning below, usage of this macro is not recommended.

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

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

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

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

1.11 Package options

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

`KeepShorthandsActive`	Tells babel not to deactivate shorthands after loading a language file, so that they are also available in the preamble.
`activeacute`	For some languages babel supports this options to set ' as a shorthand in case it is not done by default.
`activegrave`	Same for `.
`shorthands=`	⟨*char*⟩⟨*char*⟩... | off

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

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

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

safe=	none | ref | bib

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

math=	active | normal

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

config=	⟨*file*⟩

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

main=	⟨*language*⟩

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

headfoot=	⟨*language*⟩

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

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

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

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

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

strings=	generic | unicode | encoded | ⟨*label*⟩ | ⟨*font encoding*⟩

Selects the encoding of strings in languages supporting this feature. Predefined labels are `generic` (for traditional TeX, LICR and ASCII strings), `unicode` (for engines like xetex and luatex) and `encoded` (for special cases requiring mixed encodings). Other allowed values are font encoding codes (T1, T2A, LGR, L7X...), but only in languages supporting them. Be aware with encoded captions are protected, but they work in `\MakeUppercase` and the like (this feature misuses some internal LaTeX tools, so use it only as a last resort).

hyphenmap=	off | first | select | other | other*

---

[9]You can use alternatively the package silence.

New 3.9g Sets the behavior of case mapping for hyphenation, provided the language defines it.[10] It can take the following values:

off deactivates this feature and no case mapping is applied;

first sets it at the first switching commands in the current or parent scope (typically, when the aux file is first read and at \begin{document}, but also the first \selectlanguage in the preamble), and it's the default if a single language option has been stated;[11]

select sets it only at \selectlanguage;

other also sets it at otherlanguage;

other* also sets it at otherlanguage* as well as in heads and foots (if the option headfoot is used) and in auxiliary files (ie, at \select@language), and it's the default if several language options have been stated. The option first can be regarded as an optimized version of other* for monolingual documents.[12]

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

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

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 hyphenation patterns for the last language passed as option (by its name in language.dat). There are two main uses: classes and packages, and as a last resort in case there are, for some reason, incompatible languages. It can be used if you just want to select the hyphenation patterns of a single language, too.

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

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

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

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

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

---

[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 the paragraph, but if either xetex or luatex change this behavior it might be added. On the other hand, other is provided even if I [JBL] think it isn't really useful, but who knows.

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

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

1.13 `ini` **files**

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

ini files are not meant only for babel, and they has been devised as a resource for other packages. To easy interoperability between TEX and other systems, they are identified with the BCP 47 codes as preferred by the Unicode Common Language Data Repository, which was used as source for most of the data provided by these files, too (the main exception being the \...name strings).

Most of them set the date, and many also the captions (Unicode and LICR). They will be evolving with the time to add more features (something to keep in mind if backward compatibility is important). The following section shows how to make use of them currently (by means of \babelprovide), but a higher interface, based on package options, in under study. In other words, \babelprovide is mainly meant 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.

LUATEX/XETEX

```
\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}
```

NOTE The ini files just define and set some parameters, but the corresponding behavior is not always implemented. Also, there are some limitations in the engines. A few remarks follows:

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

Hebrew Niqqud marks seem to work in both engines, but cantillation marks are misplaced (xetex or luatex with Harfbuzz seems better, but still problematic).

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

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

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

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

```
\babelprovide[import,hyphenrules=+]{lao}
\babelpatterns[lao]{1 n 1ɯ 1ə 1ŋ 1ɲ 1ʔ} % Random
```

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

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

NOTE Wikipedia defines a *locale* as follows: "In computing, a locale is a set of parameters that defines the user's language, region and any special variant preferences that the user wants to see in their user interface. Usually a locale identifier consists of at least a language code and a country/region code." Babel is moving gradually from the old and fuzzy concept of *language* to the more modern of *locale*. Note each locale is by itself a separate "language", which explains why there are so many files. This is on purpose, so that possible variants can be created and/or redefined easily.

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

af	Afrikaans[ul]	az-Latn	Azerbaijani
agq	Aghem	az	Azerbaijani[ul]
ak	Akan	bas	Basaa
am	Amharic[ul]	be	Belarusian[ul]
ar	Arabic[ul]	bem	Bemba
ar-DZ	Arabic[ul]	bez	Bena
ar-MA	Arabic[ul]	bg	Bulgarian[ul]
ar-SY	Arabic[ul]	bm	Bambara
as	Assamese	bn	Bangla[ul]
asa	Asu	bo	Tibetan[u]
ast	Asturian[ul]	brx	Bodo
az-Cyrl	Azerbaijani	bs-Cyrl	Bosnian

bs-Latn	Bosnian[ul]	gu	Gujarati
bs	Bosnian[ul]	guz	Gusii
ca	Catalan[ul]	gv	Manx
ce	Chechen	ha-GH	Hausa
cgg	Chiga	ha-NE	Hausa[l]
chr	Cherokee	ha	Hausa
ckb	Central Kurdish	haw	Hawaiian
cop	Coptic	he	Hebrew[ul]
cs	Czech[ul]	hi	Hindi[u]
cu	Church Slavic	hr	Croatian[ul]
cu-Cyrs	Church Slavic	hsb	Upper Sorbian[ul]
cu-Glag	Church Slavic	hu	Hungarian[ul]
cy	Welsh[ul]	hy	Armenian[u]
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
es	Spanish[ul]	kn	Kannada[ul]
et	Estonian[ul]	ko	Korean
eu	Basque[ul]	kok	Konkani
ewo	Ewondo	ks	Kashmiri
fa	Persian[ul]	ksb	Shambala
ff	Fulah	ksf	Bafia
fi	Finnish[ul]	ksh	Colognian
fil	Filipino	kw	Cornish
fo	Faroese	ky	Kyrgyz
fr	French[ul]	lag	Langi
fr-BE	French[ul]	lb	Luxembourgish
fr-CA	French[ul]	lg	Ganda
fr-CH	French[ul]	lkt	Lakota
fr-LU	French[ul]	ln	Lingala
fur	Friulian[ul]	lo	Lao[ul]
fy	Western Frisian	lrc	Northern Luri
ga	Irish[ul]	lt	Lithuanian[ul]
gd	Scottish Gaelic[ul]	lu	Luba-Katanga
gl	Galician[ul]	luo	Luo
gsw	Swiss German	luy	Luyia

Code	Language	Code	Language
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
pl	Polish[ul]	tk	Turkmen[ul]
pms	Piedmontese[ul]	to	Tongan
ps	Pashto	tr	Turkish[ul]
pt-BR	Portuguese[ul]	twq	Tasawaq
pt-PT	Portuguese[ul]	tzm	Central Atlas Tamazight
pt	Portuguese[ul]	ug	Uyghur
qu	Quechua	uk	Ukrainian[ul]
rm	Romansh[ul]	ur	Urdu[ul]
rn	Rundi	uz-Arab	Uzbek
ro	Romanian[ul]	uz-Cyrl	Uzbek
rof	Rombo	uz-Latn	Uzbek
ru	Russian[ul]	uz	Uzbek
rw	Kinyarwanda	vai-Latn	Vai
rwk	Rwa	vai-Vaii	Vai
sa-Beng	Sanskrit	vai	Vai
sa-Deva	Sanskrit	vi	Vietnamese[ul]
sa-Gujr	Sanskrit	vun	Vunjo
sa-Knda	Sanskrit	wae	Walser
sa-Mlym	Sanskrit	xog	Soga
sa-Telu	Sanskrit	yav	Yangben

yi	Yiddish	zh-Hans-SG	Chinese
yo	Yoruba	zh-Hans	Chinese
yue	Cantonese	zh-Hant-HK	Chinese
zgh	Standard Moroccan	zh-Hant-MO	Chinese
	Tamazight	zh-Hant	Chinese
zh-Hans-HK	Chinese	zh	Chinese
zh-Hans-MO	Chinese	zu	Zulu

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

aghem	brazilian
akan	breton
albanian	british
american	bulgarian
amharic	burmese
arabic	canadian
arabic-algeria	cantonese
arabic-DZ	catalan
arabic-morocco	centralatlastamazight
arabic-MA	centralkurdish
arabic-syria	chechen
arabic-SY	cherokee
armenian	chiga
assamese	chinese-hans-hk
asturian	chinese-hans-mo
asu	chinese-hans-sg
australian	chinese-hans
austrian	chinese-hant-hk
azerbaijani-cyrillic	chinese-hant-mo
azerbaijani-cyrl	chinese-hant
azerbaijani-latin	chinese-simplified-hongkongsarchina
azerbaijani-latn	chinese-simplified-macausarchina
azerbaijani	chinese-simplified-singapore
bafia	chinese-simplified
bambara	chinese-traditional-hongkongsarchina
basaa	chinese-traditional-macausarchina
basque	chinese-traditional
belarusian	chinese
bemba	churchslavic
bena	churchslavic-cyrs
bengali	churchslavic-oldcyrillic[13]
bodo	churchsslavic-glag
bosnian-cyrillic	churchsslavic-glagolitic
bosnian-cyrl	colognian
bosnian-latin	cornish
bosnian-latn	croatian
bosnian	czech

---

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

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

Modifying and adding values to ini **files**

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

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

1.14   Selecting fonts

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

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

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

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

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

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

Babel takes care of the font language and the font script when languages are selected (as
well as the writing direction); see the recognized languages above. In most cases, you will

---

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

not need *font-options*, which is the same as in fontspec, but you may add further key/value pairs if necessary.

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

```
\documentclass{article}

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

\babelprovide[import]{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, in case it is not detected correctly. You may also pass some options to fontspec: with silent, the warnings about unavailable scripts or languages are not shown (they are only really useful when the document format is being set up).

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

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

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

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

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

This is *not* and error. This warning is shown by fontspec, not by babel. It could be irrelevant for English, but not for many other languages, including Urdu and Turkish. This is a useful and harmless warning, and if everything is fine with your document the best thing you can do is just to ignore it altogether.

TROUBLESHOOTING *Package babel Info: The following fonts are not babel standard families.*

This is *not* and error. babel assumes that if you are using `\babelfont` for a family, very likely you want to define the rest of them. If you don't, you can find some inconsistencies between families. This checking is done at the beginning of the document, at a point where we cannot know which families will be used.

Actually, there is no real need to use `\babelfont` in a monolingual document, if you set the language system in `\setmainfont` (or not, depending on what you want).

As the message explains, *there is nothing intrinsically wrong* with not defining all the families. In fact, there is nothing intrinsically wrong with not using `\babelfont` at all. But you must be aware that this may lead to some problems.

1.15   Modifying a language

Modifying the behavior of a language (say, the chapter "caption"), is sometimes necessary, but not always trivial.

- The old way, still valid for many languages, to redefine a caption is the following:

```
\addto\captionsenglish{%
  \renewcommand\contentsname{Foo}%
}
```

As of 3.15, there is no need to hide spaces with % (babel removes them), but it is advisable to do so.

- The new way, which is found in `bulgarian`, `azerbaijani`, `spanish`, `french`, `turkish`, `icelandic`, `vietnamese` and a few more, as well as in languages created with `\babelprovide` and its key `import`, is:

```
    \renewcommand\spanishchaptername{Foo}
```

- Macros to be run when a language is selected can be add to \extras⟨*lang*⟩:

```
    \addto\extrasrussian{\mymacro}
```

There is a counterpart for code to be run when a language is unselected:
\noextras⟨*lang*⟩.

NOTE Do *not* redefine a caption in the following way:

```
    \AtBeginDocument{\renewcommand\contentsname{Foo}}
```

The changes may be discarded with a language selector, and the original value restored.

NOTE These macros (\captions⟨*lang*⟩, \extras⟨*lang*⟩) may be redefined, but *must not* be used as such – they just pass information to babel, which executes them in the proper context.

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

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

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

1.16   Creating a language

New 3.10   And what if there is no style for your language or none fits your needs? You
may then define quickly a language with the help of the following macro in the preamble
(which may be used to modify an existing language, too, as explained in the previous
subsection).

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

If the language ⟨*language-name*⟩ has not been loaded as class or package option and there
are no ⟨*options*⟩, it creates an "empty" one with some defaults in its internal structure: the
hyphen rules, if not available, are set to the current ones, left and right hyphen mins are
set to 2 and 3. In either case, caption, date and language system are not defined.
If no ini file is imported with import, ⟨*language-name*⟩ is still relevant because in such a
case the hyphenation and like breaking rules (including those for South East Asian and
CJK) are based on it as provided in the ini file corresponding to that name; the same
applies to OpenType language and script.
Conveniently, some options allow to fill the language, and babel warns you about what to
do if there is a missing string. Very likely you will find alerts like that in the log file:

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

EXAMPLE Locales with names based on BCP 47 codes can be created with something like:

```
\babelprovide[import=en-US]{enUS}
```

Note, however, mixing ways to identify locales can lead to problems. For example, is `yi` the name of the language spoken by the Yi people or is it the code for Yiddish?

The main language is not changed (`danish` in this example). So, you must add `\selectlanguage{arhinish}` or other selectors where necessary.
If the language has been loaded as an argument in `\documentclass` or `\usepackage`, then `\babelprovide` redefines the requested data.

import= 〈*language-tag*〉

New 3.13 Imports data from an `ini` file, including captions, 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.
New 3.23 It may be used without a value. In such a case, the `ini` file set in the corresponding `babel-<language>.tex` (where `<language>` is the last argument in `\babelprovide`) is imported. See the list of recognized languages above. So, the previous example 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 (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. If fontspec does not define it, then babel sets its tag to that provided by the ini file. This value is particularly important because it sets the writing direction, so you must use it if for some reason the default value is wrong.

language=   ⟨*language-name*⟩

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

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

onchar=   ids | fonts

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

<dl>
<dt>mapfont=</dt>
<dd>

direction

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

</dd>
</dl>

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

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

intrapenalty= ⟨*penalty*⟩

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

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

1.17 Digits and counters

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

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

Languages providing native digits in all or some variants are:

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

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

New 4.41   Many 'ini' locale files has been extended with information about non-positional numerical systems, based on those predefined in CSS. They only work with xetex and luatex and are fully expendable (even inside an `\edef`). Currently, they are limited to numbers below 10000.

There are several ways to use them (for the availabe styles in each language, see the list below):

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

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

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

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

The styles are:

Ancient Greek `lower.ancient`, `upper.ancient`
Arabic `abjad`, `maghrebi.abjad`
Belarusan, Bulgarian, Macedonian, Serbian `lower`, `upper`
Hebrew `letters` (neither geresh nor gershayim yet)
Hindi `alphabetic`
Armenian `lower`, `upper`
Japanese `hiragana`, `hiragana.iroha`, `katakana`, `katakana.iroha`, `circled.katakana`, `informal`, `formal`, `cjk-earthly-branch`, `cjk-heavenly-stem`, `fullwidth.lower.alpha`, `fullwidth.upper.alpha`
Georgian `letters`
Greek `lower.modern`, `upper.modern`, `lower.ancient`, `upper.ancient` (all with keraia)
Khmer `consonant`
Korean `consonant`, `syllabe`, `hanja.informal`, `hanja.formal`, `hangul.formal`, `cjk-earthly-branch`, `cjk-heavenly-stem`, `fullwidth.lower.alpha`, `fullwidth.upper.alpha`
Persian `abjad`, `alphabetic`
Russian `lower`, `lower.full`, `upper`, `upper.full`
Tamil `ancient`
Thai `alphabetic`
Ukrainian `lower`, `lower.full`, `upper`, `upper.full`
Chinese `cjk-earthly-branch`, `cjk-heavenly-stem`, `fullwidth.lower.alpha`, `fullwidth.upper.alpha`

1.18   Accessing language info

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

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

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

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

`\localeinfo`	{⟨*field*⟩}

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

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

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

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

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

the macro `\hechap` will contain the string פרק.
Babel remembers which `ini` files have been loaded. There is a loop named `\LocaleForEach` to traverse the list, where #1 is the name of the current item, so that `\LocaleForEach{\message{ **#1** }}` just shows the loaded `ini`'s.

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

1.19   Hyphenation and line breaking

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

New 3.37-3.39 With luatex it is now possible to define non-standard hyphenation rules, like f-f → ff-f, repeated hyphens, ranked ruled (or more precisely, 'penalized' hyphenation points), and so on. No rules are currently provided by default, but they can be defined as shown in the following example, where {1} is the first captured char (between ( ) in the pattern):

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

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

This feature is activated with the first \babelposthyphenation.

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

EXAMPLE Although the main purpose of this command is non-standard hyphenation, it may actually be used for other transformations (after hyphenation is applied, so you must take discretionaries into account). For example, you can use the string replacement to replace a character (or series of them) by another character (or series of them). Thus, to enter *ž* as zh and *š* as sh in a newly created locale for transliterated Russian:

---

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

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

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

1.20   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.[16]
Some languages sharing the same script define macros to switch it (eg, `\textcyrillic`), but be aware they may also set the language to a certain default. Even the babel core defined `\textlatin`, but is was somewhat buggy because in some cases it messed up encodings and fonts (for example, if the main Latin encoding was LY1), and therefore it has been deprecated.[17]

`\ensureascii`   {⟨*text*⟩}

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

1.21   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

---

[16]The so-called Unicode fonts do not improve the situation either. So, a font suited for Vietnamese is not necessarily suited for, say, the romanization of Indic languages, and the fact it contains glyphs for Modern Greek does not mean it includes them for Classic Greek.
[17]But still defined for backwards compatibility.

setting bidi text has many subtleties (see for example
<https://www.w3.org/TR/html-bidi/>). A basic stable version for other engines must
wait. This applies to text; there is a basic support for **graphical** elements, including the
`picture` environment (with pict2e) and pfg/tikz. Also, indexes and the like are under
study, as well as math (there is progress in the latter, too, but for example `cases` may
fail).

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

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

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

There are some package options controlling bidi writing.

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

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

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

```
\documentclass{article}

\usepackage[bidi=basic]{babel}

\babelprovide[import, main]{arabic}

\babelfont{rm}{FreeSerif}

\begin{document}

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

\end{document}
```

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

```
\documentclass{book}

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

\babelprovide[onchar=ids fonts]{arabic}

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

\begin{document}

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

\end{document}
```

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

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

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

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

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

New 3.16 *To be expanded.* Selects which layout elements are adapted in bidi documents, including some text elements (except with options loading the bidi package, which provides its own mechanism to control these elements). You may use several options with a dot-separated list (eg, layout=counters.contents.sectioning). This list will be expanded in future releases. Note not all options are required by all engines.

sectioning makes sure the sectioning macros are typeset in the main language, but with the title text in the current language (see below \BabelPatchSection for further details).
counters required in all engines (except luatex with bidi=basic) to reorder section numbers and the like (eg, ⟨*subsection*⟩.⟨*section*⟩); required in xetex and pdftex for counters in general, as well as in luatex with bidi=default; required in luatex for

38

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.[18]

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

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

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

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

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

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

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

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

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

EXAMPLE Typically, in an Arabic document you would need:

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

`\babelsublr` {⟨*lr-text*⟩}

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

---

[18]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

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

The same name can be applied to several events. Hooks may be enabled and disabled for
all defined events with \EnableBabelHook{⟨*name*⟩}, \DisableBabelHook{⟨*name*⟩}.
Names containing the string babel are reserved (they are used, for example, by
\useshortands* to add a hook for the event afterextras). New 3.33 They may be also
applied to a specific language with the optional argument; language-specific settings are
executed after global ones.

Current events are the following; in some of them you can use one to three TeX parameters
(#1, #2, #3), with the meaning given:

adddialect (language name, dialect name) Used by luababel.def to load the patterns if
not preloaded.

patterns (language name, language with encoding) Executed just after the \language has
been set. The second argument has the patterns name actually selected (in the form of
either lang:ENC or lang).

hyphenation (language name, language with encoding) Executed locally just before
exceptions given in \babelhyphenation are actually set.

defaultcommands Used (locally) in \StartBabelCommands.

encodedcommands (input, font encodings) Used (locally) in \StartBabelCommands. Both
xetex and luatex make sure the encoded text is read correctly.

stopcommands Used to reset the above, if necessary.

write This event comes just after the switching commands are written to the aux file.

beforeextras Just before executing \extras⟨*language*⟩. This event and the next one
should not contain language-dependent code (for that, add it to \extras⟨*language*⟩).

afterextras Just after executing \extras⟨*language*⟩. For example, the following
deactivates shorthands in all languages:

```
\AddBabelHook{noshort}{afterextras}{\languageshorthands{none}}
```

stringprocess Instead of a parameter, you can manipulate the macro \BabelString
containing the string to be defined with \SetString. For example, to use an expanded
version of the string in the definition, write:

```
\AddBabelHook{myhook}{stringprocess}{%
  \protected@edef\BabelString{\BabelString}}
```

`initiateactive` (char as active, char as other, original char) New 3.9i Executed just
after a shorthand has been 'initiated'. The three parameters are the same character
with different catcodes: active, other (`\string`'ed) and the original one.

`afterreset` New 3.9i Executed when selecting a language just after `\originalTeX` is
run and reset to its base value, before executing `\captions`⟨*language*⟩ and
`\date`⟨*language*⟩.

Four events are used in hyphen.cfg, which are handled in a quite different way for
efficiency reasons – unlike the precedent ones, they only have a single hook and replace a
default definition.

`everylanguage` (language) Executed before every language patterns are loaded.
`loadkernel` (file) By default just defines a few basic commands. It can be used to define
different versions of them or to load a file.
`loadpatterns` (patterns file) Loads the patterns file. Used by luababel.def.
`loadexceptions` (exceptions file) Loads the exceptions file. Used by luababel.def.

`\BabelContentsFiles` New 3.9a This macro contains a list of "toc" types requiring a command to switch the
language. Its default value is toc,lof,lot, but you may redefine it with `\renewcommand`
(it's up to you to make sure no toc type is duplicated).

1.24 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 indonesian, bahasa, indon, bahasai
Interlingua interlingua

Irish Gaelic irish
Italian italian
Latin latin
Lower Sorbian lowersorbian
Malay malay, melayu, bahasam
North Sami samin
Norwegian norsk, nynorsk
Polish polish
Portuguese portuguese, portuges[19], 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 or luatexja). For example, if you have got the velthuis/devnag package, you can create a file with extension `.dn`:

```
\documentclass{article}
\usepackage[hindi]{babel}
\begin{document}
{\dn devaanaa.m priya.h}
\end{document}
```

Then you preprocess it with devnag ⟨*file*⟩, which creates ⟨*file*⟩`.tex`; you can then typeset the latter with LaTeX.

1.25 Unicode character properties in luatex

New 3.32   Part of the babel job is to apply Unicode rules to some script-specific features based on some properties. Currently, they are 3, namely, direction (ie, bidi class), mirroring glyphs, and line breaking for CJK scripts. These properties are stored in lua tables, which you can modify with the following macro (for example, to set them for glyphs in the PUA).

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

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

---

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

43

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

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

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

1.26 Tweaking some features

\babeladjust   {⟨*key-value-list*⟩}

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

1.27 Tips, workarounds, known issues and notes

- If you use the document class book *and* you use `\ref` inside the argument of `\chapter` (or just use `\ref` inside `\MakeUppercase`), LaTeX will keep complaining about an undefined label. To prevent such problems, you 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.[20] So, if you write a chunk of French text with `\foreinglanguage`, the apostrophes might not be taken into account. This is a limitation of TeX, not of babel. Alternatively, you may use `\useshorthands` to activate `'` and `\defineshorthand`, or redefine `\textquoteright` (the latter is called by the non-ASCII right quote).

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

- 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.

1.28   Current and future work

The current work is focused on the so-called complex scripts in luatex. In 8-bit engines, babel provided a basic support for bidi text as part of the style for Hebrew, but it is somewhat unsatisfactory and internally replaces some hardwired commands by other hardwired commands (generic changes would be much better).
Useful additions would be, for example, time, currency, addresses and personal names.[21]. But that is the easy part, because they don't require modifying the LaTeX internals. Calendars (Arabic, Persian, Indic, etc.) are under study.
Also interesting are differences in the sentence structure or related to it. For example, in Basque the number precedes the name (including chapters), in Hungarian "from (1)" is "(1)-ből", but "from (3)" is "(3)-ból", in Spanish an item labelled "3.º" may be referred to as either "ítem 3.º" or "3.ᵉʳ ítem", and so on.
An option to manage bidirectional document layout in luatex (lists, footnotes, etc.) is almost finished, but xetex required more work. Unfortunately, proper support for xetex requires patching somehow lots of macros and packages (and some issues related to

---

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

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

`\specials` remain, like color and hyperlinks), so babel resorts to the bidi package (by Vafa Khalighi). See the babel repository for a small example (`xe-bidi`).

1.29 Tentative and experimental code

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

Old and deprecated stuff
A couple of tentative macros were provided by babel (≥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}
```

2 Loading languages with `language.dat`

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

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

2.1 Format

In that file the person who maintains a TeX environment has to record for which languages he has hyphenation patterns *and* in which files these are stored[24]. When hyphenation exceptions are stored in a separate file this can be indicated by naming that file *after* the file with the hyphenation patterns.
The file can contain empty lines and comments, as well as lines which start with an equals (=) sign. Such a line will instruct LaTeX that the hyphenation patterns just processed have to be known under an alternative name. Here is an example:

---

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

---

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

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

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

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

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

Some recommendations:

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

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

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

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

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

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

3.1 Guidelines for contributed languages

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.

---

[26]But not removed, for backward compatibility.

- 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 you need further assistance and technical advice in the development of language styles, I am willing to help you. And of course, you can make any suggestion you like.

3.2 Basic macros

In the core of the babel system, several macros are defined for use in language definition files. Their purpose is to make a new language known. The first two are related to hyphenation patterns.

`\addlanguage` The macro `\addlanguage` is a non-outer version of the macro `\newlanguage`, defined in `plain.tex` version 3.x. For older versions of `plain.tex` and `lplain.tex` a substitute definition is used. Here "language" is used in the TeX sense of set of hyphenation patterns.

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

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

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

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

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

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

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

`\extras`⟨*lang*⟩ The macro `\extras`⟨*lang*⟩ contains all the extra definitions needed for a specific language. This macro, like the following, is a hook – you can add things to it, but it must not be used directly.

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

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

`\main@language` To postpone the activation of the definitions needed for a language until the beginning of a document, all language definition files should use `\main@language` instead of

	\selectlanguage. This will just store the name of the language, and the proper language will be activated at the start of the document.
\ProvidesLanguage	The macro \ProvidesLanguage should be used to identify the language definition files. Its syntax is similar to the syntax of the LaTeX command \ProvidesPackage.
\LdfInit	The macro \LdfInit performs a couple of standard checks that must be made at the beginning of a language definition file, such as checking the category code of the @-sign, preventing the .ldf file from being processed twice, etc.
\ldf@quit	The macro \ldf@quit does work needed if a .ldf file was processed earlier. This includes resetting the category code of the @-sign, preparing the language to be activated at \begin{document} time, and ending the input stream.
\ldf@finish	The macro \ldf@finish does work needed at the end of each .ldf file. This includes resetting the category code of the @-sign, loading a local configuration file, and preparing the language to be activated at \begin{document} time.
\loadlocalcfg	After processing a language definition file, LaTeX can be instructed to load a local configuration file. This file can, for instance, be used to add strings to \captions⟨*lang*⟩ to support local document classes. The user will be informed that this configuration file has been loaded. This macro is called by \ldf@finish.
\substitutefontfamily	(Deprecated.) This command takes three arguments, a font encoding and two font family names. It creates a font description file for the first font in the given encoding. This .fd file will instruct LaTeX to use a font from the second family when a font from the first family in the given encoding seems to be needed.

3.3 Skeleton

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

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

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

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

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

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

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

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

\StartBabelCommands*{<dialect>}{captions}
\SetString\chaptername{<chapter name>}
```

```
% More strings

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

\EndBabelCommands

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

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

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

```
\AtEndOfPackage{%
  \RequirePackage{dingbat}%        Delay package
  \savebox{\myeye}{\eye}}%         And direct usage
\newsavebox{\myeye}
\newcommand\myanchor{\anchor}%     But OK inside command
```

3.4 Support for active characters

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

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

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

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

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

3.5 Support for saving macro definitions

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

We provide two macros for this[27].

\babel@save   To save the current meaning of any control sequence, the macro \babel@save is provided. It takes one argument, ⟨*csname*⟩, the control sequence for which the meaning has to be saved.

\babel@savevariable   A second macro is provided to save the current value of a variable. In this context, anything that is allowed after the \the primitive is considered to be a variable. The macro takes one argument, the ⟨*variable*⟩.

The effect of the preceding macros is to append a piece of code to the current definition of \originalTeX. When \originalTeX is expanded, this code restores the previous definition of the control sequence or the previous value of the variable.

3.6 Support for extending macros

\addto   The macro \addto{⟨*control sequence*⟩}{⟨*TEX code*⟩} can be used to extend the definition of a macro. The macro need not be defined (ie, it can be undefined or \relax). This macro can, for instance, be used in adding instructions to a macro like \extrasenglish.

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

3.7 Macros common to a number of languages

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

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

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

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

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

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

3.8 Encoding-dependent strings

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

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

---

[27]This mechanism was introduced by Bernd Raichle.

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

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

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

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

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

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

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

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

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

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

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

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

\EndBabelCommands
```

A real example is:

---

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

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

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

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

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

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

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

\EndBabelCommands
```

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

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

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

\EndBabelCommands Marks the end of the series of blocks.

\AfterBabelCommands {⟨*code*⟩}

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

---

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

54

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

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

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

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

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

#1 is replaced by the roman numeral.

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

Sets globally code to be executed at \MakeUppercase and \MakeLowercase. The code would typically be things like \let\BB\bb and \uccode or \lccode (although for the reasons explained above, changes in lc/uc codes may not work). A ⟨*map-list*⟩ is a series of macros using the internal format of \@uclclist (eg, \bb\BB\cc\CC). The mandatory arguments take precedence over the optional one. This command, unlike \SetString, is executed always (even without strings), and it is intended for minor readjustments only. For example, as T1 is the default case mapping in LaTeX, we 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.

55

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

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

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

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

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

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

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

4   Changes

4.1   Changes in babel version 3.9

Most of the changes in version 3.9 were related to bugs, either to fix them (there were lots), or to provide some alternatives. Even new features like \babelhyphen are intended to solve a certain problem (in this case, the lacking of a uniform syntax and behavior for shorthands across languages). These changes are described in this manual in the corresponding place. A selective list follows:

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

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

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

- ' (with activeacute) had the original value when writing to an auxiliary file, and things like an infinite loop 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 following description is no longer valid, because switch and plain have been merged into babel.def.
The babel package after unpacking consists of the following files:

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

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

6   `locale` **directory**

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

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

Keys may be further qualified in a particular language with a suffix starting with an 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=3.42.1989⟩⟩
2 ⟨⟨date=2020/04/26⟩⟩
```

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

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

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

\bbl@add@list   This internal macro adds its second argument to a comma separated list in its first argument. When the list is not defined yet (or empty), it will be initiated. It presumes expandable character strings.

```
21 \def\bbl@add@list#1#2{%
22   \edef#1{%
23     \bbl@ifunset{\bbl@stripslash#1}%
24       {}%
25       {\ifx#1\@empty\else#1,\fi}%
26     #2}}
```

\bbl@afterelse   Because the code that is used in the handling of active characters may need to look ahead,
\bbl@afterfi   we take extra care to 'throw' it over the \else and \fi parts of an \if-statement[30]. These macros will break if another \if...\fi statement appears in one of the arguments and it is not enclosed in braces.

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

---

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

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

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

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

```
35 \def\bbl@tempa#1{%
36   \long\def\bbl@trim##1##2{%
37     \futurelet\bbl@trim@a\bbl@trim@c##2\@nil\@nil##1\@nil\relax{##1}}%
38   \def\bbl@trim@c{%
39     \ifx\bbl@trim@a\@sptoken
40       \expandafter\bbl@trim@b
41     \else
42       \expandafter\bbl@trim@b\expandafter#1%
43     \fi}%
44   \long\def\bbl@trim@b#1##1 \@nil{\bbl@trim@i##1}}
45 \bbl@tempa{ }
46 \long\def\bbl@trim@i#1\@nil#2\relax#3{#3{#1}}
47 \long\def\bbl@trim@def#1{\bbl@trim{\def#1}}
```

\bbl@ifunset To check if a macro is defined, we create a new macro, which does the same as \@ifundefined. However, in an ϵ-tex engine, it is based on \ifcsname, which is more efficient, and do not waste memory.

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

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

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

59

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).

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

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

\bbl@replace

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

An extensison to the previous macro. It takes into account the parameters, and it is string based (ie, if you replace elax by ho, then \relax becomes \rho). No checking is done at all, because it is not a general purpose macro, and it is used by babel only when it works (an example where it does *not* work is in \bbl@TG@@date, and also fails if there are macros with spaces, because they are retokenized). It may change! (or even merged with \bbl@replace; I'm not sure ckecking the replacement is really necessary or just paranoia).

```
103 \ifx\detokenize\@undefined\else % Unused macros if old Plain TeX
104   \bbl@exp{\def\\\bbl@parsedef##1\detokenize{macro:}}#2->#3\relax{%
105     \def\bbl@tempa{#1}%
106     \def\bbl@tempb{#2}%
107     \def\bbl@tempe{#3}}
108   \def\bbl@sreplace#1#2#3{%
109     \begingroup
110       \expandafter\bbl@parsedef\meaning#1\relax
111       \def\bbl@tempc{#2}%
112       \edef\bbl@tempc{\expandafter\strip@prefix\meaning\bbl@tempc}%
113       \def\bbl@tempd{#3}%
```

60

```
114      \edef\bbl@tempd{\expandafter\strip@prefix\meaning\bbl@tempd}%
115      \bbl@xin@{\bbl@tempc}{\bbl@tempe}% If not in macro, do nothing
116      \ifin@
117        \bbl@exp{\\\bbl@replace\\\bbl@tempe{\bbl@tempc}{\bbl@tempd}}%
118        \def\bbl@tempc{%      Expanded an executed below as 'uplevel'
119           \\\makeatletter % "internal" macros with @ are assumed
120           \\\scantokens{%
121              \bbl@tempa\\\@namedef{\bbl@stripslash#1}\bbl@tempb{\bbl@tempe}}%
122           \catcode64=\the\catcode64\relax}%  Restore @
123      \else
124        \let\bbl@tempc\@empty  % Not \relax
125      \fi
126      \bbl@exp{%      For the 'uplevel' assignments
127      \endgroup
128        \bbl@tempc}}  % empty or expand to set #1 with changes
129 \fi
```

Two further tools. `\bbl@samestring` first expand its arguments and then compare their expansion (sanitized, so that the catcodes do not matter). `\bbl@engine` takes the following values: 0 is pdfTeX, 1 is luatex, and 2 is xetex. You may use the latter it in your language style if you want.

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

```
153 ⟨⟨*Make sure ProvidesFile is defined⟩⟩ ≡
154 \ifx\ProvidesFile\@undefined
155   \def\ProvidesFile#1[#2 #3 #4]{%
156     \wlog{File: #1 #4 #3 <#2>}%
157     \let\ProvidesFile\@undefined}
158 \fi
159 ⟨⟨/Make sure ProvidesFile is defined⟩⟩
```

7.1 Multiple languages

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

```
160 ⟨⟨∗Define core switching macros⟩⟩ ≡
161 \ifx\language\@undefined
162   \csname newcount\endcsname\language
163 \fi
164 ⟨⟨/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.

```
165 ⟨⟨∗Define core switching macros⟩⟩ ≡
166 \ifx\newlanguage\@undefined
167   \csname newcount\endcsname\last@language
168   \def\addlanguage#1{%
169     \global\advance\last@language\@ne
170     \ifnum\last@language<\@cclvi
171     \else
172       \errmessage{No room for a new \string\language!}%
173     \fi
174     \global\chardef#1\last@language
175     \wlog{\string#1 = \string\language\the\last@language}}
176 \else
177   \countdef\last@language=19
178   \def\addlanguage{\alloc@9\language\chardef\@cclvi}
179 \fi
180 ⟨⟨/Define core switching macros⟩⟩
```

Now we make sure all required files are loaded. When the command `\AtBeginDocument` doesn't exist we assume that we are dealing with a plain-based format or LaTeX2.09. In that case the file `plain.def` is needed (which also defines `\AtBeginDocument`, and therefore it is not loaded twice). We need the first part when the format is created, and `\orig@dump` is used as a flag. Otherwise, we need to use the second part, so `\orig@dump` is not defined (`plain.def` undefines it).
Check if the current version of `switch.def` has been previously loaded (mainly, `hyphen.cfg`). If not, load it now. We cannot load `babel.def` here because we first need to declare and process the package options.

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

This file also takes care of a number of compatibility issues with other packages an defines a few aditional package options. Apart from all the language options below we also have a few options that influence the behavior of language definition files.

Many of the following options don't do anything themselves, they are just defined in order to make it possible for babel and language definition files to check if one of them was specified by the user.

The first two options are for debugging.

```
181 ⟨∗package⟩
182 \NeedsTeXFormat{LaTeX2e}[2005/12/01]
183 \ProvidesPackage{babel}[⟨⟨date⟩⟩ ⟨⟨version⟩⟩ The Babel package]
184 \@ifpackagewith{babel}{debug}
185   {\providecommand\bbl@trace[1]{\message{^^J[ #1 ]}}%
186    \let\bbl@debug\@firstofone}
187   {\providecommand\bbl@trace[1]{}%
188    \let\bbl@debug\@gobble}
189 ⟨⟨Basic macros⟩⟩
190   % Temporarily repeat here the code for errors
191   \def\bbl@error#1#2{%
192     \begingroup
193       \def\\{\MessageBreak}%
194       \PackageError{babel}{#1}{#2}%
195     \endgroup}
196   \def\bbl@warning#1{%
197     \begingroup
198       \def\\{\MessageBreak}%
199       \PackageWarning{babel}{#1}%
200     \endgroup}
201   \def\bbl@infowarn#1{%
202     \begingroup
203       \def\\{\MessageBreak}%
204       \GenericWarning
205         {(babel) \@spaces\@spaces\@spaces}%
206         {Package babel Info: #1}%
207     \endgroup}
208   \def\bbl@info#1{%
209     \begingroup
210       \def\\{\MessageBreak}%
211       \PackageInfo{babel}{#1}%
212     \endgroup}
213     \def\bbl@nocaption{\protect\bbl@nocaption@i}
214 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
215   \global\@namedef{#2}{\textbf{?#1?}}%
216   \@nameuse{#2}%
217   \bbl@warning{%
218     \@backslashchar#2 not set. Please, define\\%
219     it in the preamble with something like:\\%
220     \string\renewcommand\@backslashchar#2{..}\\%
221     Reported}}
222 \def\bbl@tentative{\protect\bbl@tentative@i}
223 \def\bbl@tentative@i#1{%
224   \bbl@warning{%
225     Some functions for '#1' are tentative.\\%
226     They might not work as expected and their behavior\\%
227     could change in the future.\\%
228     Reported}}
229 \def\@nolanerr#1{%
230   \bbl@error
231     {You haven't defined the language #1\space yet.\\%
232      Perhaps you misspelled it or your installation\\%
233      is not complete}%
234     {Your command will be ignored, type <return> to proceed}}
```

```
235 \def\@nopatterns#1{%
236   \bbl@warning
237     {No hyphenation patterns were preloaded for\\%
238      the language `#1' into the format.\\%
239      Please, configure your TeX system to add them and\\%
240      rebuild the format. Now I will use the patterns\\%
241      preloaded for \bbl@nulllanguage\space instead}}
242     % End of errors
243 \@ifpackagewith{babel}{silent}
244   {\let\bbl@info\@gobble
245    \let\bbl@infowarn\@gobble
246    \let\bbl@warning\@gobble}
247   {}
248 %
249 \def\AfterBabelLanguage#1{%
250   \global\expandafter\bbl@add\csname#1.ldf-h@@k\endcsname}%
```

If the format created a list of loaded languages (in \bbl@languages), get the name of the 0-th to show the actual language used. Also avaliable with base, because it just shows info.

```
251 \ifx\bbl@languages\@undefined\else
252   \begingroup
253     \catcode`\^^I=12
254     \@ifpackagewith{babel}{showlanguages}{%
255       \begingroup
256         \def\bbl@elt#1#2#3#4{\wlog{#2^^I#1^^I#3^^I#4}}%
257         \wlog{<*languages>}%
258         \bbl@languages
259         \wlog{</languages>}%
260       \endgroup}{}
261   \endgroup
262   \def\bbl@elt#1#2#3#4{%
263     \ifnum#2=\z@
264       \gdef\bbl@nulllanguage{#1}%
265       \def\bbl@elt##1##2##3##4{}%
266     \fi}%
267   \bbl@languages
268 \fi%
```

7.3 base

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

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

```
269 \bbl@trace{Defining option 'base'}
270 \@ifpackagewith{babel}{base}{%
271   \let\bbl@onlyswitch\@empty
272   \let\bbl@provide@locale\relax
273   \input babel.def
274   \let\bbl@onlyswitch\@undefined
275   \ifx\directlua\@undefined
276     \DeclareOption*{\bbl@patterns{\CurrentOption}}%
277   \else
278     \input luababel.def
279     \DeclareOption*{\bbl@patterns@lua{\CurrentOption}}%
280   \fi
```

```
281  \DeclareOption{base}{}%
282  \DeclareOption{showlanguages}{}%
283  \ProcessOptions
284  \global\expandafter\let\csname opt@babel.sty\endcsname\relax
285  \global\expandafter\let\csname ver@babel.sty\endcsname\relax
286  \global\let\@ifl@ter@@\@ifl@ter
287  \def\@ifl@ter#1#2#3#4#5{\global\let\@ifl@ter\@ifl@ter@@}%
288  \endinput}{}%
289 % \end{macrocode}
290 %
291 % TODO. Code for lua bidi options must be moved to a logical place. The
292 % problem is |\RequirePackage|, which is forbidden allowed in the options
293 % section.
294 %
295 % \begin{macrocode}
296 \ifodd\bbl@engine
297   \def\bbl@activate@preotf{%
298     \let\bbl@activate@preotf\relax  % only once
299     \directlua{
300       Babel = Babel or {}
301       %
302       function Babel.pre_otfload_v(head)
303         if Babel.numbers and Babel.digits_mapped then
304           head = Babel.numbers(head)
305         end
306         if Babel.bidi_enabled then
307           head = Babel.bidi(head, false, dir)
308         end
309         return head
310       end
311       %
312       function Babel.pre_otfload_h(head, gc, sz, pt, dir)
313         if Babel.numbers and Babel.digits_mapped then
314           head = Babel.numbers(head)
315         end
316         if Babel.bidi_enabled then
317           head = Babel.bidi(head, false, dir)
318         end
319         return head
320       end
321       %
322       luatexbase.add_to_callback('pre_linebreak_filter',
323         Babel.pre_otfload_v,
324         'Babel.pre_otfload_v',
325         luatexbase.priority_in_callback('pre_linebreak_filter',
326           'luaotfload.node_processor') or nil)
327       %
328       luatexbase.add_to_callback('hpack_filter',
329         Babel.pre_otfload_h,
330         'Babel.pre_otfload_h',
331         luatexbase.priority_in_callback('hpack_filter',
332           'luaotfload.node_processor') or nil)
333     }}
334   \let\bbl@tempa\relax
335   \@ifpackagewith{babel}{bidi=basic}%
336     {\def\bbl@tempa{basic}}%
337     {\@ifpackagewith{babel}{bidi=basic-r}%
338       {\def\bbl@tempa{basic-r}}%
339       {}}
```

```
340   \ifx\bbl@tempa\relax\else
341     \let\bbl@beforeforeign\leavevmode
342     \AtEndOfPackage{\EnableBabelHook{babel-bidi}}%
343     \RequirePackage{luatexbase}%
344     \directlua{
345       require('babel-data-bidi.lua')
346       require('babel-bidi-\bbl@tempa.lua')
347     }
348     \bbl@activate@preotf
349   \fi
350 \fi
```

7.4 `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.

```
351 \bbl@trace{key=value and another general options}
352 \bbl@csarg\let{tempa\expandafter}\csname opt@babel.sty\endcsname
353 \def\bbl@tempb#1.#2{%
354     #1\ifx\@empty#2\else,\bbl@afterfi\bbl@tempb#2\fi}%
355 \def\bbl@tempd#1.#2\@nnil{%
356   \ifx\@empty#2%
357     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
358   \else
359     \in@{=}{#1}\ifin@
360       \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.#2}%
361     \else
362       \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
363       \bbl@csarg\edef{mod@#1}{\bbl@tempb#2}%
364     \fi
365   \fi}
366 \let\bbl@tempc\@empty
367 \bbl@foreach\bbl@tempa{\bbl@tempd#1.\@empty\@nnil}
368 \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.

```
369 \DeclareOption{KeepShorthandsActive}{}
370 \DeclareOption{activeacute}{}
371 \DeclareOption{activegrave}{}
372 \DeclareOption{debug}{}
373 \DeclareOption{noconfigs}{}
374 \DeclareOption{showlanguages}{}
375 \DeclareOption{silent}{}
376 \DeclareOption{mono}{}
377 \DeclareOption{shorthands=off}{\bbl@tempa shorthands=\bbl@tempa}
378 % Don't use. Experimental. TODO.
379 \newif\ifbbl@single
380 \DeclareOption{selectors=off}{\bbl@singletrue}
381 ⟨⟨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.

```
382 \let\bbl@opt@shorthands\@nnil
383 \let\bbl@opt@config\@nnil
384 \let\bbl@opt@main\@nnil
385 \let\bbl@opt@headfoot\@nnil
386 \let\bbl@opt@layout\@nnil
```

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

```
387 \def\bbl@tempa#1=#2\bbl@tempa{%
388   \bbl@csarg\ifx{opt@#1}\@nnil
389     \bbl@csarg\edef{opt@#1}{#2}%
390   \else
391     \bbl@error
392     {Bad option `#1=#2'. Either you have misspelled the\\%
393      key or there is a previous setting of `#1'. Valid\\%
394      keys are, among others, `shorthands', `main', `bidi',\\%
395      `strings', `config', `headfoot', `safe', `math'.}%
396     {See the manual for further details.}
397   \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.

```
398 \let\bbl@language@opts\@empty
399 \DeclareOption*{%
400   \bbl@xin@{\string=}{\CurrentOption}%
401   \ifin@
402     \expandafter\bbl@tempa\CurrentOption\bbl@tempa
403   \else
404     \bbl@add@list\bbl@language@opts{\CurrentOption}%
405   \fi}
```

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

```
406 \ProcessOptions*
```

7.5   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=....

```
407 \bbl@trace{Conditional loading of shorthands}
408 \def\bbl@sh@string#1{%
409   \ifx#1\@empty\else
410     \ifx#1t\string~%
411     \else\ifx#1c\string,%
412     \else\string#1%
413     \fi\fi
414     \expandafter\bbl@sh@string
415   \fi}
416 \ifx\bbl@opt@shorthands\@nnil
417   \def\bbl@ifshorthand#1#2#3{#2}%
418 \else\ifx\bbl@opt@shorthands\@empty
```

```
419    \def\bbl@ifshorthand#1#2#3{#3}%
420 \else
```

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

```
421    \def\bbl@ifshorthand#1{%
422      \bbl@xin@{\string#1}{\bbl@opt@shorthands}%
423      \ifin@
424        \expandafter\@firstoftwo
425      \else
426        \expandafter\@secondoftwo
427      \fi}
```

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

```
428    \edef\bbl@opt@shorthands{%
429      \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.

```
430    \bbl@ifshorthand{'}%
431      {\PassOptionsToPackage{activeacute}{babel}}{}
432    \bbl@ifshorthand{`}%
433      {\PassOptionsToPackage{activegrave}{babel}}{}
434 \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.

```
435 \ifx\bbl@opt@headfoot\@nnil\else
436   \g@addto@macro\@resetactivechars{%
437      \set@typeset@protect
438      \expandafter\select@language@x\expandafter{\bbl@opt@headfoot}%
439      \let\protect\noexpand}
440 \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.

```
441 \ifx\bbl@opt@safe\@undefined
442   \def\bbl@opt@safe{BR}
443 \fi
444 \ifx\bbl@opt@main\@nnil\else
445   \edef\bbl@language@opts{%
446      \ifx\bbl@language@opts\@empty\else\bbl@language@opts,\fi
447        \bbl@opt@main}
448 \fi
```

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

```
449 \bbl@trace{Defining IfBabelLayout}
450 \ifx\bbl@opt@layout\@nnil
451   \newcommand\IfBabelLayout[3]{#3}%
452 \else
453   \newcommand\IfBabelLayout[1]{%
454      \@expandtwoargs\in@{.#1.}{.\bbl@opt@layout.}%
455      \ifin@
456        \expandafter\@firstoftwo
457      \else
458        \expandafter\@secondoftwo
459      \fi}
460 \fi
```

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

```
461 \input babel.def
```

7.6   Cross referencing macros

The LATEX book states:

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

When the above quote should still be true when a document is typeset in a language that has active characters, special care has to be taken of the category codes of these characters when they appear in an argument of the cross referencing macros.
When a cross referencing command processes its argument, all tokens in this argument should be character tokens with category 'letter' or 'other'.
The following package options control which macros are to be redefined.

```
462 ⟨⟨∗More package options⟩⟩ ≡
463 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
464 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
465 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
466 ⟨⟨/More package options⟩⟩
```

\@newl@bel First we open a new group to keep the changed setting of \protect local and then we set the @safe@actives switch to true to make sure that any shorthand that appears in any of the arguments immediately expands to its non-active self.

```
467 \bbl@trace{Cross referencing macros}
468 \ifx\bbl@opt@safe\@empty\else
469   \def\@newl@bel#1#2#3{%
470     {\@safe@activestrue
471      \bbl@ifunset{#1@#2}%
472         \relax
473         {\gdef\@multiplelabels{%
474            \@latex@warning@no@line{There were multiply-defined labels}}%
475          \@latex@warning@no@line{Label `#2' multiply defined}}%
476     \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.

```
477   \CheckCommand*\@testdef[3]{%
478     \def\reserved@a{#3}%
479     \expandafter\ifx\csname#1@#2\endcsname\reserved@a
480     \else
481       \@tempswatrue
482     \fi}
```

Now that we made sure that \@testdef still has the same definition we can rewrite it. First we make the shorthands 'safe'. Then we use \bbl@tempa as an 'alias' for the macro that contains the label which is being checked. Then we define \bbl@tempb just as \@newl@bel does it. When the label is defined we replace the definition of \bbl@tempa by its meaning. If the label didn't change, \bbl@tempa and \bbl@tempb should be identical macros.

```
483   \def\@testdef#1#2#3{%  TODO. With @samestring?
484     \@safe@activestrue
485     \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
486     \def\bbl@tempb{#3}%
487     \@safe@activesfalse
```

```
488    \ifx\bbl@tempa\relax
489    \else
490      \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
491    \fi
492    \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
493    \ifx\bbl@tempa\bbl@tempb
494    \else
495      \@tempswatrue
496    \fi}
497 \fi
```

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

```
498 \bbl@xin@{R}\bbl@opt@safe
499 \ifin@
500   \bbl@redefinerobust\ref#1{%
501     \@safe@activestrue\org@ref{#1}\@safe@activesfalse}
502   \bbl@redefinerobust\pageref#1{%
503     \@safe@activestrue\org@pageref{#1}\@safe@activesfalse}
504 \else
505   \let\org@ref\ref
506   \let\org@pageref\pageref
507 \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.

```
508 \bbl@xin@{B}\bbl@opt@safe
509 \ifin@
510   \bbl@redefine\@citex[#1]#2{%
511     \@safe@activestrue\edef\@tempa{#2}\@safe@activesfalse
512     \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.

```
513   \AtBeginDocument{%
514     \@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.)

```
515     \def\@citex[#1][#2]#3{%
516       \@safe@activestrue\edef\@tempa{#3}\@safe@activesfalse
517       \org@@citex[#1][#2]{\@tempa}}%
518     }{}}
```

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

```
519   \AtBeginDocument{%
520     \@ifpackageloaded{cite}{%
521       \def\@citex[#1]#2{%
```

```
522        \@safe@activestrue\org@@citex[#1]{#2}\@safe@activesfalse}%
523      }{}}
```

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

```
524  \bbl@redefine\nocite#1{%
525     \@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.

```
526  \bbl@redefine\bibcite{%
527     \bbl@cite@choice
528     \bibcite}
```

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

```
529  \def\bbl@bibcite#1#2{%
530     \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.

```
531  \def\bbl@cite@choice{%
532     \global\let\bibcite\bbl@bibcite
533     \@ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{}%
534     \@ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{}%
535     \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.

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

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

```
537  \bbl@redefine\@bibitem#1{%
538     \@safe@activestrue\org@@bibitem{#1}\@safe@activesfalse}
539 \else
540   \let\org@nocite\nocite
541   \let\org@@citex\@citex
542   \let\org@bibcite\bibcite
543   \let\org@@bibitem\@bibitem
544 \fi
```

7.7 Marks

\markright Because the output routine is asynchronous, we must pass the current language attribute to the head lines. To achieve this we need to adapt the definition of \markright and \markboth somewhat. However, headlines and footlines can contain text outside marks; for that we must take some actions in the output routine if the 'headfoot' options is used.

We need to make some redefinitions to the output routine to avoid an endless loop and to correctly handle the page number in bidi documents.

```
545 \bbl@trace{Marks}
546 \IfBabelLayout{sectioning}
547   {\ifx\bbl@opt@headfoot\@nnil
548     \g@addto@macro\@resetactivechars{%
549       \set@typeset@protect
550       \expandafter\select@language@x\expandafter{\bbl@main@language}%
551       \let\protect\noexpand
552       \edef\thepage{% TODO. Only with bidi. See also above
553         \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}}%
554    \fi}
555   {\ifbbl@single\else
556     \bbl@ifunset{markright }\bbl@redefine\bbl@redefinerobust
557     \markright#1{%
558       \bbl@ifblank{#1}%
559         {\org@markright{}}%
560         {\toks@{#1}%
561          \bbl@exp{%
562            \\\org@markright{\\\protect\\\foreignlanguage{\languagename}%
563              {\\\protect\\\bbl@restore@actives\the\toks@}}}}}%
```

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

```
564     \ifx\@mkboth\markboth
565       \def\bbl@tempc{\let\@mkboth\markboth}
566     \else
567       \def\bbl@tempc{}
568     \fi
569     \bbl@ifunset{markboth }\bbl@redefine\bbl@redefinerobust
570     \markboth#1#2{%
571       \protected@edef\bbl@tempb##1{%
572         \protect\foreignlanguage
573         {\languagename}{\protect\bbl@restore@actives##1}}%
574       \bbl@ifblank{#1}%
575         {\toks@{}}%
576         {\toks@\expandafter{\bbl@tempb{#1}}}%
577       \bbl@ifblank{#2}%
578         {\@temptokena{}}%
579         {\@temptokena\expandafter{\bbl@tempb{#2}}}%
580       \bbl@exp{\\\org@markboth{\the\toks@}{\the\@temptokena}}}
581       \bbl@tempc
582    \fi}  % end ifbbl@single, end \IfBabelLayout
```

7.8 Preventing clashes with other packages

7.8.1 ifthen

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

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

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

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

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

```
583 \bbl@trace{Preventing clashes with other packages}
584 \bbl@xin@{R}\bbl@opt@safe
585 \ifin@
586   \AtBeginDocument{%
587     \@ifpackageloaded{ifthen}{%
588       \bbl@redefine@long\ifthenelse#1#2#3{%
589         \let\bbl@temp@pref\pageref
590         \let\pageref\org@pageref
591         \let\bbl@temp@ref\ref
592         \let\ref\org@ref
593         \@safe@activestrue
594         \org@ifthenelse{#1}%
595           {\let\pageref\bbl@temp@pref
596            \let\ref\bbl@temp@ref
597            \@safe@activesfalse
598            #2}%
599           {\let\pageref\bbl@temp@pref
600            \let\ref\bbl@temp@ref
601            \@safe@activesfalse
602            #3}%
603       }%
604     }{}%
605   }
```

7.8.2 varioref

\@@vpageref
\vrefpagenum
\Ref

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

```
606   \AtBeginDocument{%
607     \@ifpackageloaded{varioref}{%
608       \bbl@redefine\@@vpageref#1[#2]#3{%
609         \@safe@activestrue
610         \org@@@vpageref{#1}[#2]{#3}%
611         \@safe@activesfalse}%
612       \bbl@redefine\vrefpagenum#1#2{%
613         \@safe@activestrue
614         \org@vrefpagenum{#1}{#2}%
615         \@safe@activesfalse}%
```

The package varioref defines \Ref to be a robust command wich uppercases the first character of the reference text. In order to be able to do that it needs to access the expandable form of \ref. So we employ a little trick here. We redefine the (internal)

command \Ref␣ to call \org@ref instead of \ref. The disadvantage of this solution is that whenever the definition of \Ref changes, this definition needs to be updated as well.

```
616        \expandafter\def\csname Ref \endcsname#1{%
617          \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
618        }{}%
619      }
620 \fi
```

7.8.3 hhline

\hhline Delaying the activation of the shorthand characters has introduced a problem with the hhline package. The reason is that it uses the ':' character which is made active by the french support in babel. Therefore we need to *reload* the package when the ':' is an active character. Note that this happens *after* the category code of the @-sign has been changed to other, so we need to temporarily change it to letter again.

```
621 \AtEndOfPackage{%
622   \AtBeginDocument{%
623     \@ifpackageloaded{hhline}%
624       {\expandafter\ifx\csname normal@char\string:\endcsname\relax
625        \else
626          \makeatletter
627          \def\@currname{hhline}\input{hhline.sty}\makeatother
628        \fi}%
629      {}}}
```

7.8.4 hyperref

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

```
630 \AtBeginDocument{%
631   \ifx\pdfstringdefDisableCommands\@undefined\else
632     \pdfstringdefDisableCommands{\languageshorthands{system}}%
633   \fi}
```

7.8.5 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.

```
634 \DeclareRobustCommand{\FOREIGNLANGUAGE}[1]{%
635   \lowercase{\foreignlanguage{#1}}}
```

\substitutefontfamily The command \substitutefontfamily creates an .fd file on the fly. The first argument is an encoding mnemonic, the second and third arguments are font family names. This command is deprecated. Use the tools provides by LaTeX.

```
636 \def\substitutefontfamily#1#2#3{%
637   \lowercase{\immediate\openout15=#1#2.fd\relax}%
638   \immediate\write15{%
639     \string\ProvidesFile{#1#2.fd}%
640     [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
641      \space generated font description file]^^J
642     \string\DeclareFontFamily{#1}{#2}{}^^J
```

```
643    \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{}^^J
644    \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{}^^J
645    \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{}^^J
646    \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{}^^J
647    \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{}^^J
648    \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{}^^J
649    \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{}^^J
650    \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{}^^J
651    }%
652  \closeout15
653  }
654 \@onlypreamble\substitutefontfamily
```

7.9 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

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

75

```
689    \fi
690  \fi}
```

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

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

```
691 \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.

```
692 \AtBeginDocument{%
693   \@ifpackageloaded{fontspec}%
694     {\xdef\latinencoding{%
695        \ifx\UTFencname\@undefined
696          EU\ifcase\bbl@engine\or2\or1\fi
697        \else
698          \UTFencname
699        \fi}}%
700     {\gdef\latinencoding{OT1}%
701      \ifx\cf@encoding\bbl@t@one
702        \xdef\latinencoding{\bbl@t@one}%
703      \else
704        \ifx\@fontenc@load@list\@undefined
705          \@ifl@aded{def}{t1enc}{\xdef\latinencoding{\bbl@t@one}}{}%
706        \else
707          \def\@elt#1{,#1,}%
708          \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
709          \let\@elt\relax
710          \bbl@xin@{,T1,}\bbl@tempa
711          \ifin@
712            \xdef\latinencoding{\bbl@t@one}%
713          \fi
714        \fi
715      \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.

```
716 \DeclareRobustCommand{\latintext}{%
717   \fontencoding{\latinencoding}\selectfont
718   \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.

```
719 \ifx\@undefined\DeclareTextFontCommand
720   \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
721 \else
722   \DeclareTextFontCommand{\textlatin}{\latintext}
723 \fi
```

7.10 Basic bidi support

Work in progress. This code is currently placed here for practical reasons. It will be moved to the correct place soon, I hope.

It is loosely based on `rlbabel.def`, but most of it has been developed from scratch. This babel module (by Johannes Braams and Boris Lavva) has served the purpose of typesetting R documents for two decades, and despite its flaws I think it is still a good starting point (some parts have been copied here almost verbatim), partly thanks to its simplicity. I've also looked at ARABI (by Youssef Jabri), which is compatible with babel.

There are two ways of modifying macros to make them "bidi", namely, by patching the internal low-level macros (which is what I have done with lists, columns, counters, tocs, much like `rlbabel` did), and by introducing a "middle layer" just below the user interface (sectioning, footnotes).

- pdftex provides a minimal support for bidi text, and it must be done by hand. Vertical typesetting is not possible.

- xetex is somewhat better, thanks to its font engine (even if not always reliable) and a few additional tools. However, very little is done at the paragraph level. Another challenging problem is text direction does not honour TeX grouping.

- luatex can provide the most complete solution, as we can manipulate almost freely the node list, the generated lines, and so on, but bidi text does not work out of the box and some development is necessary. It also provides tools to properly set left-to-right and right-to-left page layouts. As LuaTeX-ja shows, vertical typesetting is possible, too.

```
724 \bbl@trace{Basic (internal) bidi support}
725 \def\bbl@alscripts{,Arabic,Syriac,Thaana,}
726 \def\bbl@rscripts{%
727   ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
728   Old Hungarian,Old Hungarian,Lydian,Mandaean,Manichaean,%
729   Manichaean,Meroitic Cursive,Meroitic,Old North Arabian,%
730   Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
731   Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
732   Old South Arabian,}%
733 \def\bbl@provide@dirs#1{%
734   \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
735   \ifin@
736     \global\bbl@csarg\chardef{wdir@#1}\@ne
737     \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
738     \ifin@
739       \global\bbl@csarg\chardef{wdir@#1}\tw@  % useless in xetex
740     \fi
741   \else
742     \global\bbl@csarg\chardef{wdir@#1}\z@
743   \fi
744   \ifodd\bbl@engine
745     \bbl@csarg\ifcase{wdir@#1}%
746       \directlua{ Babel.locale_props[\the\localeid].textdir = 'l' }%
747     \or
748       \directlua{ Babel.locale_props[\the\localeid].textdir = 'r' }%
749     \or
750       \directlua{ Babel.locale_props[\the\localeid].textdir = 'al' }%
751     \fi
752   \fi}
753 \def\bbl@switchdir{%
754   \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
755   \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
```

```
756    \bbl@exp{\\\bbl@setdirs\bbl@cl{wdir}}}
757 \def\bbl@setdirs#1{% TODO - math
758    \ifcase\bbl@select@type % TODO - strictly, not the right test
759      \bbl@bodydir{#1}%
760      \bbl@pardir{#1}%
761    \fi
762    \bbl@textdir{#1}}
763 \ifodd\bbl@engine  % luatex=1
764    \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
765    \DisableBabelHook{babel-bidi}
766    \chardef\bbl@thetextdir\z@
767    \chardef\bbl@thepardir\z@
768    \def\bbl@getluadir#1{%
769      \directlua{
770        if tex.#1dir == 'TLT' then
771          tex.sprint('0')
772        elseif tex.#1dir == 'TRT' then
773          tex.sprint('1')
774        end}}
775    \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
776      \ifcase#3\relax
777        \ifcase\bbl@getluadir{#1}\relax\else
778          #2 TLT\relax
779        \fi
780      \else
781        \ifcase\bbl@getluadir{#1}\relax
782          #2 TRT\relax
783        \fi
784      \fi}
785    \def\bbl@textdir#1{%
786      \bbl@setluadir{text}\textdir{#1}%
787      \chardef\bbl@thetextdir#1\relax
788      \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*3+#1}}
789    \def\bbl@pardir#1{%
790      \bbl@setluadir{par}\pardir{#1}%
791      \chardef\bbl@thepardir#1\relax}
792    \def\bbl@bodydir{\bbl@setluadir{body}\bodydir}
793    \def\bbl@pagedir{\bbl@setluadir{page}\pagedir}
794    \def\bbl@dirparastext{\pardir\the\textdir\relax}%    %%%%
795    % Sadly, we have to deal with boxes in math with basic.
796    % Activated every math with the package option bidi=:
797    \def\bbl@mathboxdir{%
798      \ifcase\bbl@thetextdir\relax
799        \everyhbox{\textdir TLT\relax}%
800      \else
801        \everyhbox{\textdir TRT\relax}%
802      \fi}
803 \else % pdftex=0, xetex=2
804    \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
805    \DisableBabelHook{babel-bidi}
806    \newcount\bbl@dirlevel
807    \chardef\bbl@thetextdir\z@
808    \chardef\bbl@thepardir\z@
809    \def\bbl@textdir#1{%
810      \ifcase#1\relax
811        \chardef\bbl@thetextdir\z@
812        \bbl@textdir@i\beginL\endL
813      \else
814        \chardef\bbl@thetextdir\@ne
```

```
815        \bbl@textdir@i\beginR\endR
816      \fi}
817  \def\bbl@textdir@i#1#2{%
818    \ifhmode
819      \ifnum\currentgrouplevel>\z@
820        \ifnum\currentgrouplevel=\bbl@dirlevel
821          \bbl@error{Multiple bidi settings inside a group}%
822            {I'll insert a new group, but expect wrong results.}%
823          \bgroup\aftergroup#2\aftergroup\egroup
824        \else
825          \ifcase\currentgrouptype\or % 0 bottom
826            \aftergroup#2% 1 simple {}
827          \or
828            \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
829          \or
830            \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
831          \or\or\or % vbox vtop align
832          \or
833            \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
834          \or\or\or\or\or\or % output math disc insert vcent mathchoice
835          \or
836            \aftergroup#2% 14 \begingroup
837          \else
838            \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
839          \fi
840        \fi
841        \bbl@dirlevel\currentgrouplevel
842      \fi
843      #1%
844    \fi}
845  \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
846  \let\bbl@bodydir\@gobble
847  \let\bbl@pagedir\@gobble
848  \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).

```
849  \def\bbl@xebidipar{%
850    \let\bbl@xebidipar\relax
851    \TeXXeTstate\@ne
852    \def\bbl@xeeverypar{%
853      \ifcase\bbl@thepardir
854        \ifcase\bbl@thetextdir\else\beginR\fi
855      \else
856        {\setbox\z@\lastbox\beginR\box\z@}%
857      \fi}%
858    \let\bbl@severypar\everypar
859    \newtoks\everypar
860    \everypar=\bbl@severypar
861    \bbl@severypar{\bbl@xeeverypar\the\everypar}}
862  \def\bbl@tempb{%
863    \let\bbl@textdir@i\@gobbletwo
864    \let\bbl@xebidipar\@empty
865    \AddBabelHook{bidi}{foreign}{%
866      \def\bbl@tempa{\def\BabelText########1}%
867      \ifcase\bbl@thetextdir
868        \expandafter\bbl@tempa\expandafter{\BabelText{\LR{####1}}}%
869      \else
```

79

```
870        \expandafter\bbl@tempa\expandafter{\BabelText{\RL{####1}}}%
871      \fi}
872    \def\bbl@pardir##1{\ifcase##1\relax\setLR\else\setRL\fi}}
873  \@ifpackagewith{babel}{bidi=bidi}{\bbl@tempb}{}%
874  \@ifpackagewith{babel}{bidi=bidi-l}{\bbl@tempb}{}%
875  \@ifpackagewith{babel}{bidi=bidi-r}{\bbl@tempb}{}%
876 \fi
```

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

```
877 \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbl@textdir\z@#1}}
878 \AtBeginDocument{%
879  \ifx\pdfstringdefDisableCommands\@undefined\else
880    \ifx\pdfstringdefDisableCommands\relax\else
881      \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
882    \fi
883  \fi}
```

7.11 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.

```
884 \bbl@trace{Local Language Configuration}
885 \ifx\loadlocalcfg\@undefined
886  \@ifpackagewith{babel}{noconfigs}%
887    {\let\loadlocalcfg\@gobble}%
888    {\def\loadlocalcfg#1{%
889      \InputIfFileExists{#1.cfg}%
890        {\typeout{*************************************^^J%
891                  * Local config file #1.cfg used^^J%
892                  *}}%
893        \@empty}}
894 \fi
```

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

```
895 \ifx\@unexpandable@protect\@undefined
896  \def\@unexpandable@protect{\noexpand\protect\noexpand}
897  \long\def\protected@write#1#2#3{%
898    \begingroup
899      \let\thepage\relax
900      #2%
901      \let\protect\@unexpandable@protect
902      \edef\reserved@a{\write#1{#3}}%
903      \reserved@a
904    \endgroup
905    \if@nobreak\ifvmode\nobreak\fi\fi}
906 \fi
907 %
908 % \subsection{Language options}
909 %
910 %    Languages are loaded when processing the corresponding option
911 %    \textit{except} if a |main| language has been set. In such a
912 %    case, it is not loaded until all options has been processed.
```

```
913 %    The following macro inputs the ldf file and does some additional
914 %    checks (|\input| works, too, but possible errors are not catched).
915 %
916 %    \begin{macrocode}
917 \bbl@trace{Language options}
918 \let\bbl@afterlang\relax
919 \let\BabelModifiers\relax
920 \let\bbl@loaded\@empty
921 \def\bbl@load@language#1{%
922   \InputIfFileExists{#1.ldf}%
923     {\edef\bbl@loaded{\CurrentOption
924       \ifx\bbl@loaded\@empty\else,\bbl@loaded\fi}%
925     \expandafter\let\expandafter\bbl@afterlang
926       \csname\CurrentOption.ldf-h@@k\endcsname
927     \expandafter\let\expandafter\BabelModifiers
928       \csname bbl@mod@\CurrentOption\endcsname}%
929   {\bbl@error{%
930     Unknown option `\CurrentOption'. Either you misspelled it\\%
931     or the language definition file \CurrentOption.ldf was not found}{%
932     Valid options are: shorthands=, KeepShorthandsActive,\\%
933     activeacute, activegrave, noconfigs, safe=, main=, math=\\%
934     headfoot=, strings=, config=, hyphenmap=, or a language name.}}}
```

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

```
935 \def\bbl@try@load@lang#1#2#3{%
936   \IfFileExists{\CurrentOption.ldf}%
937     {\bbl@load@language{\CurrentOption}}%
938     {#1\bbl@load@language{#2}#3}}
939 \DeclareOption{afrikaans}{\bbl@try@load@lang{}{dutch}{}}
940 \DeclareOption{brazil}{\bbl@try@load@lang{}{portuges}{}}
941 \DeclareOption{brazilian}{\bbl@try@load@lang{}{portuges}{}}
942 \DeclareOption{hebrew}{%
943   \input{rlbabel.def}%
944   \bbl@load@language{hebrew}}
945 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}{}}
946 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}{}}
947 \DeclareOption{nynorsk}{\bbl@try@load@lang{}{norsk}{}}
948 \DeclareOption{polutonikogreek}{%
949   \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}
950 \DeclareOption{portuguese}{\bbl@try@load@lang{}{portuges}{}}
951 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}{}}
952 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}{}}
953 \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.

```
954 \ifx\bbl@opt@config\@nnil
955   \@ifpackagewith{babel}{noconfigs}{}%
956     {\InputIfFileExists{bblopts.cfg}%
957       {\typeout{*************************************^^J%
958               * Local config file bblopts.cfg used^^J%
959              *}}%
960     {}}%
961 \else
962   \InputIfFileExists{\bbl@opt@config.cfg}%
963     {\typeout{*************************************^^J%
```

81

```
964            * Local config file \bbl@opt@config.cfg used^^J%
965            *}}%
966     {\bbl@error{%
967        Local config file `\bbl@opt@config.cfg' not found}{%
968        Perhaps you misspelled it.}}%
969 \fi
```

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

```
970 \bbl@for\bbl@tempa\bbl@language@opts{%
971   \bbl@ifunset{ds@\bbl@tempa}%
972     {\edef\bbl@tempb{%
973        \noexpand\DeclareOption
974          {\bbl@tempa}%
975          {\noexpand\bbl@load@language{\bbl@tempa}}}%
976      \bbl@tempb}%
977      \@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.

```
978 \bbl@foreach\@classoptionslist{%
979   \bbl@ifunset{ds@#1}%
980     {\IfFileExists{#1.ldf}%
981        {\DeclareOption{#1}{\bbl@load@language{#1}}}%
982        {}}%
983     {}}
```

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

```
984 \ifx\bbl@opt@main\@nnil\else
985   \expandafter
986   \let\expandafter\bbl@loadmain\csname ds@\bbl@opt@main\endcsname
987   \DeclareOption{\bbl@opt@main}{}
988 \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):

```
989 \def\AfterBabelLanguage#1{%
990   \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang}{}}
991 \DeclareOption*{}
992 \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.

```
993 \bbl@trace{Option 'main'}
994 \ifx\bbl@opt@main\@nnil
995   \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}
996   \let\bbl@tempc\@empty
997   \bbl@for\bbl@tempb\bbl@tempa{%
998     \bbl@xin@{,\bbl@tempb,}{,\bbl@loaded,}%
```

```
999      \ifin@\edef\bbl@tempc{\bbl@tempb}\fi}
1000  \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
1001  \expandafter\bbl@tempa\bbl@loaded,\@nnil
1002  \ifx\bbl@tempb\bbl@tempc\else
1003      \bbl@warning{%
1004          Last declared language option is `\bbl@tempc',\\%
1005          but the last processed one was `\bbl@tempb'.\\%
1006          The main language cannot be set as both a global\\%
1007          and a package option. Use `main=\bbl@tempc' as\\%
1008          option. Reported}%
1009    \fi
1010 \else
1011    \DeclareOption{\bbl@opt@main}{\bbl@loadmain}
1012    \ExecuteOptions{\bbl@opt@main}
1013    \DeclareOption*{}
1014    \ProcessOptions*
1015 \fi
1016 \def\AfterBabelLanguage{%
1017    \bbl@error
1018      {Too late for \string\AfterBabelLanguage}%
1019      {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.

```
1020 \ifx\bbl@main@language\@undefined
1021    \bbl@info{%
1022      You haven't specified a language. I'll use 'nil'\\%
1023      as the main language. Reported}
1024      \bbl@load@language{nil}
1025 \fi
1026 ⟨/package⟩
1027 ⟨∗core⟩
```

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

The kernel of the babel system is currently stored in `babel.def`. The file `babel.def`
contains most of the code. The file `hyphen.cfg` is a file that can be loaded into the format,
which is necessary when you want to be able to switch hyphenation patterns.
Because plain TeX users might want to use some of the features of the babel system too,
care has to be taken that plain TeX can process the files. For this reason the current format
will have to be checked in a number of places. Some of the code below is common to plain
TeX and LaTeX, some of it is for the LaTeX case only.
Plain formats based on etex (etex, xetex, luatex) don't load `hyphen.cfg` but `etex.src`,
which follows a different naming convention, so we need to define the babel names. It
presumes `language.def` exists and it is the same file used when formats were created.

8.1   Tools

```
1028 \ifx\ldf@quit\@undefined\else
1029 \endinput\fi % Same line!
1030 ⟨⟨Make sure ProvidesFile is defined⟩⟩
1031 \ProvidesFile{babel.def}[⟨⟨date⟩⟩ ⟨⟨version⟩⟩ Babel common definitions]
```

The file `babel.def` expects some definitions made in the LaTeX 2ε style file. So, In LaTeX2.09
and Plain we must provide at least some predefined values as well some tools to set them
(even if not all options are available). There are no package options, and therefore and

alternative mechanism is provided. For the moment, only \babeloptionstrings and \babeloptionmath are provided, which can be defined before loading babel. \BabelModifiers can be set too (but not sure it works).

```
1032 \ifx\AtBeginDocument\@undefined  % TODO. change test.
1033 ⟨⟨Emulate LaTeX⟩⟩
1034 \def\languagename{english}%
1035 \let\bbl@opt@shorthands\@nnil
1036 \def\bbl@ifshorthand#1#2#3{#2}%
1037 \let\bbl@language@opts\@empty
1038 \ifx\babeloptionstrings\@undefined
1039   \let\bbl@opt@strings\@nnil
1040 \else
1041   \let\bbl@opt@strings\babeloptionstrings
1042 \fi
1043 \def\BabelStringsDefault{generic}
1044 \def\bbl@tempa{normal}
1045 \ifx\babeloptionmath\bbl@tempa
1046   \def\bbl@mathnormal{\noexpand\textormath}
1047 \fi
1048 \def\AfterBabelLanguage#1#2{}
1049 \ifx\BabelModifiers\@undefined\let\BabelModifiers\relax\fi
1050 \let\bbl@afterlang\relax
1051 \def\bbl@opt@safe{BR}
1052 \ifx\@uclclist\@undefined\let\@uclclist\@empty\fi
1053 \ifx\bbl@trace\@undefined\def\bbl@trace#1{}\fi
1054 \expandafter\newif\csname ifbbl@single\endcsname
1055 \fi
```

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

```
1056 \ifx\bbl@trace\@undefined
1057   \let\LdfInit\endinput
1058   \def\ProvidesLanguage#1{\endinput}
1059 \endinput\fi % Same line!
```

And continue.

9   Multiple languages

This is not a separate file (switch.def) anymore.

Plain TeX version 3.0 provides the primitive \language that is used to store the current language. When used with a pre-3.0 version this function has to be implemented by allocating a counter.

```
1060 ⟨⟨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.

```
1061 \def\bbl@version{⟨⟨version⟩⟩}
1062 \def\bbl@date{⟨⟨date⟩⟩}
1063 \def\adddialect#1#2{%
1064   \global\chardef#1#2\relax
1065   \bbl@usehooks{adddialect}{{#1}{#2}}%
1066   \begingroup
1067     \count@#1\relax
1068     \def\bbl@elt##1##2##3##4{%
1069       \ifnum\count@=##2\relax
1070         \bbl@info{\string#1 = using hyphenrules for ##1\\%
1071                   (\string\language\the\count@)}%
1072         \def\bbl@elt####1####2####3####4{}%
```

```
1073        \fi}%
1074    \bbl@cs{languages}%
1075  \endgroup}
```

`\bbl@iflanguage` executes code only if the language l@ exists. Otherwise raises and error. The argument of `\bbl@fixname` has to be a macro name, as it may get "fixed" if casing (lc/uc) is wrong. It's intented to fix a long-standing bug when `\foreignlanguage` and the like appear in a `\MakeXXXcase`. However, a lowercase form is not imposed to improve backward compatibility (perhaps you defined a language named MYLANG, but unfortunately mixed case names cannot be trapped). Note l@ is encapsulated, so that its case does not change.

```
1076 \def\bbl@fixname#1{%
1077  \begingroup
1078    \def\bbl@tempe{l@}%
1079    \edef\bbl@tempd{\noexpand\@ifundefined{\noexpand\bbl@tempe#1}}%
1080    \bbl@tempd
1081      {\lowercase\expandafter{\bbl@tempd}%
1082         {\uppercase\expandafter{\bbl@tempd}%
1083           \@empty
1084           {\edef\bbl@tempd{\def\noexpand#1{#1}}%
1085            \uppercase\expandafter{\bbl@tempd}}}%
1086        {\edef\bbl@tempd{\def\noexpand#1{#1}}%
1087         \lowercase\expandafter{\bbl@tempd}}}%
1088      \@empty
1089    \edef\bbl@tempd{\endgroup\def\noexpand#1{#1}}%
1090    \bbl@tempd
1091    \bbl@exp{\\\bbl@usehooks{languagename}{{\languagename}{#1}}}}
1092 \def\bbl@iflanguage#1{%
1093  \@ifundefined{l@#1}{\@nolanerr{#1}\@gobble}\@firstofone}
```

After a name has been 'fixed', the selectors will try to load the language. If even the fixed name is not defined, will load it on the fly, either based on its name, or if activated, its BCP47 code.

We first need a couple of macros for a simple BCP 47 look up. It also makes sure, with `\bbl@bcpcase`, casing is the correct one, so that sr-latn-ba becomes fr-Latn-BA. Note #4 may contain some `\@empty`'s, but they are eventually removed. `\bbl@bcplookup` either returns the found ini or it is `\relax`.

```
1094 \def\bbl@bcpcase#1#2#3#4\@@#5{%
1095  \ifx\@empty#3%
1096    \uppercase{\def#5{#1#2}}%
1097  \else
1098    \uppercase{\def#5{#1}}%
1099    \lowercase{\edef#5{#5#2#3#4}}%
1100  \fi}
1101 \def\bbl@bcplookup#1-#2-#3-#4\@@{%
1102  \let\bbl@bcp\relax
1103  \lowercase{\def\bbl@tempa{#1}}%
1104  \ifx\@empty#2%
1105    \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
1106  \else\ifx\@empty#3%
1107    \bbl@bcpcase#2\@empty\@empty\@@\bbl@tempb
1108    \IfFileExists{babel-\bbl@tempa-\bbl@tempb.ini}%
1109      {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb}}%
1110      {}%
1111    \ifx\bbl@bcp\relax
1112      \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
1113    \fi
1114  \else
```

```
1115        \bbl@bcpcase#2\@empty\@empty\@@\bbl@tempb
1116        \bbl@bcpcase#3\@empty\@empty\@@\bbl@tempc
1117        \IfFileExists{babel-\bbl@tempa-\bbl@tempb-\bbl@tempc.ini}%
1118          {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb-\bbl@tempc}}%
1119          {}%
1120        \ifx\bbl@bcp\relax
1121          \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
1122            {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
1123            {}%
1124        \fi
1125        \ifx\bbl@bcp\relax
1126          \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
1127            {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
1128            {}%
1129        \fi
1130        \ifx\bbl@bcp\relax
1131          \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
1132        \fi
1133    \fi\fi}
1134 \let\bbl@autoload@options\@empty
1135 \let\bbl@initoload\relax
1136 \def\bbl@provide@locale{%
1137    \ifx\babelprovide\@undefined
1138      \bbl@error{For a language to be defined on the fly 'base'\\%
1139                  is not enough, and the whole package must be\\%
1140                  loaded. Either delete the 'base' option or\\%
1141                  request the languages explicitly}%
1142                {See the manual for further details.}%
1143    \fi
1144 % TODO. Option to search if loaded, with \LocaleForEach
1145    \let\bbl@auxname\languagename % Still necessary. TODO
1146    \bbl@ifunset{bbl@bcp@map@\languagename}{}% Move uplevel??
1147      {\edef\languagename{\@nameuse{bbl@bcp@map@\languagename}}}%
1148    \ifbbl@bcpallowed
1149      \expandafter\ifx\csname date\languagename\endcsname\relax
1150        \expandafter
1151        \bbl@bcplookup\languagename-\@empty-\@empty-\@empty\@@
1152        \ifx\bbl@bcp\relax\else  % Returned by \bbl@bcplookup
1153          \edef\languagename{\bbl@bcp@prefix\bbl@bcp}%
1154          \edef\localename{\bbl@bcp@prefix\bbl@bcp}%
1155          \expandafter\ifx\csname date\languagename\endcsname\relax
1156            \let\bbl@initoload\bbl@bcp
1157            \bbl@exp{\\\babelprovide[\bbl@autoload@bcpoptions]{\languagename}}%
1158            \let\bbl@initoload\relax
1159          \fi
1160          \bbl@csarg\xdef{bcp@map@\bbl@bcp}{\localename}%
1161        \fi
1162      \fi
1163    \fi
1164    \expandafter\ifx\csname date\languagename\endcsname\relax
1165      \IfFileExists{babel-\languagename.tex}%
1166        {\bbl@exp{\\\babelprovide[\bbl@autoload@options]{\languagename}}}%
1167        {}%
1168    \fi}
```

\iflanguage   Users might want to test (in a private package for instance) which language is currently active. For this we provide a test macro, \iflanguage, that has three arguments. It checks whether the first argument is a known language. If so, it compares the first argument with the value of \language. Then, depending on the result of the comparison, it executes

either the second or the third argument.

```
1169 \def\iflanguage#1{%
1170   \bbl@iflanguage{#1}{%
1171     \ifnum\csname l@#1\endcsname=\language
1172       \expandafter\@firstoftwo
1173     \else
1174       \expandafter\@secondoftwo
1175     \fi}}
```

9.1   Selecting the language

\selectlanguage   The macro \selectlanguage checks whether the language is already defined before it performs its actual task, which is to update \language and activate language-specific definitions.

```
1176 \let\bbl@select@type\z@
1177 \edef\selectlanguage{%
1178   \noexpand\protect
1179   \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.

```
1180 \ifx\@undefined\protect\let\protect\relax\fi
```

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

```
1181 \let\xstring\string
```

Since version 3.5 babel writes entries to the auxiliary files in order to typeset table of contents etc. in the correct language environment.

\bbl@pop@language   *But* when the language change happens *inside* a group the end of the group doesn't write anything to the auxiliary files. Therefore we need TeX's aftergroup mechanism to help us. The command \aftergroup stores the token immediately following it to be executed when the current group is closed. So we define a temporary control sequence \bbl@pop@language to be executed at the end of the group. It calls \bbl@set@language with the name of the current language as its argument.

\bbl@language@stack   The previous solution works for one level of nesting groups, but as soon as more levels are used it is no longer adequate. For that case we need to keep track of the nested languages using a stack mechanism. This stack is called \bbl@language@stack and initially empty.

```
1182 \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:

```
1183 \def\bbl@push@language{%
1184   \ifx\languagename\@undefined\else
1185     \xdef\bbl@language@stack{\languagename+\bbl@language@stack}%
1186   \fi}
```

Retrieving information from the stack is a little bit less simple, as we need to remove the element from the stack while storing it in the macro \languagename. For this we first define a helper function.

\bbl@pop@lang This macro stores its first element (which is delimited by the '+'-sign) in \languagename and stores the rest of the string (delimited by '-') in its third argument.

```
1187 \def\bbl@pop@lang#1+#2&#3{%
1188   \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.

```
1189 \let\bbl@ifrestoring\@secondoftwo
1190 \def\bbl@pop@language{%
1191   \expandafter\bbl@pop@lang\bbl@language@stack&\bbl@language@stack
1192   \let\bbl@ifrestoring\@firstoftwo
1193   \expandafter\bbl@set@language\expandafter{\languagename}%
1194   \let\bbl@ifrestoring\@secondoftwo}
```

Once the name of the previous language is retrieved from the stack, it is fed to \bbl@set@language to do the actual work of switching everything that needs switching.

An alternative way to identify languages (in the babel sense) with a numerical value is introduced in 3.30. This is one of the first steps for a new interface based on the concept of locale, which explains the name of \localeid. This means \l@... will be reserved for hyphenation patterns (so that two locales can share the same rules).

```
1195 \chardef\localeid\z@
1196 \def\bbl@id@last{0}    % No real need for a new counter
1197 \def\bbl@id@assign{%
1198   \bbl@ifunset{bbl@id@@\languagename}%
1199     {\count@\bbl@id@last\relax
1200      \advance\count@\@ne
1201      \bbl@csarg\chardef{id@@\languagename}\count@
1202      \edef\bbl@id@last{\the\count@}%
1203      \ifcase\bbl@engine\or
1204        \directlua{
1205          Babel = Babel or {}
1206          Babel.locale_props = Babel.locale_props or {}
1207          Babel.locale_props[\bbl@id@last] = {}
1208          Babel.locale_props[\bbl@id@last].name = '\languagename'
1209        }%
1210      \fi}%
1211    {}%
1212    \chardef\localeid\bbl@cl{id@}}
```

The unprotected part of \selectlanguage.

```
1213 \expandafter\def\csname selectlanguage \endcsname#1{%
1214   \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\tw@\fi
1215   \bbl@push@language
1216   \aftergroup\bbl@pop@language
1217   \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.

```
1218 \def\BabelContentsFiles{toc,lof,lot}
1219 \def\bbl@set@language#1{% from selectlanguage, pop@
1220   % The old buggy way. Preserved for compatibility.
1221   \edef\languagename{%
1222     \ifnum\escapechar=\expandafter`\string#1\@empty
1223     \else\string#1\@empty\fi}%
1224   \ifcat\relax\noexpand#1%
1225     \expandafter\ifx\csname date\languagename\endcsname\relax
1226       \edef\languagename{#1}%
1227       \let\localename\languagename
1228     \else
1229       \bbl@info{Using '\string\language' instead of 'language' is\\%
1230                 not recommended. If what you want is to use\\%
1231                 a macro containing the actual locale, make\\%
1232                 sure it does not not match any language.\\%
1233                 Reported}%
1234 %                I'll\\%
1235 %                try to fix '\string\localename', but I cannot promise\\%
1236 %                anything. Reported}%
1237       \ifx\scantokens\@undefined
1238         \def\localename{??}%
1239       \else
1240         \scantokens\expandafter{\expandafter
1241           \def\expandafter\localename\expandafter{\languagename}}%
1242       \fi
1243     \fi
1244   \else
1245     \def\localename{#1}% This one has the correct catcodes
1246   \fi
1247   \select@language{\languagename}%
1248   % write to auxs
1249   \expandafter\ifx\csname date\languagename\endcsname\relax\else
1250     \if@filesw
1251       \ifx\babel@aux\@gobbletwo\else % Set if single in the first, redundant
1252         \protected@write\@auxout{}{\string\babel@aux{\bbl@auxname}{}}%
1253       \fi
1254       \bbl@usehooks{write}{}%
1255     \fi
1256   \fi}
1257 %
1258 \newif\ifbbl@bcpallowed
1259 \bbl@bcpallowedfalse
1260 \def\select@language#1{% from set@, babel@aux
1261   % set hymap
1262   \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
1263   % set name
1264   \edef\languagename{#1}%
1265   \bbl@fixname\languagename
1266   % TODO. name@map must be here?
1267   \bbl@provide@locale
1268   \bbl@iflanguage\languagename{%
1269     \expandafter\ifx\csname date\languagename\endcsname\relax
1270       \bbl@error
1271         {Unknown language `\languagename'. Either you have\\%
1272          misspelled its name, it has not been installed,\\%
```

```
1273         or you requested it in a previous run. Fix its name,\\%
1274         install it or just rerun the file, respectively. In\\%
1275         some cases, you may need to remove the aux file}%
1276       {You may proceed, but expect wrong results}%
1277    \else
1278      % set type
1279      \let\bbl@select@type\z@
1280      \expandafter\bbl@switch\expandafter{\languagename}%
1281    \fi}}
1282 \def\babel@aux#1#2{%
1283  \select@language{#1}%
1284  \bbl@foreach\BabelContentsFiles{%
1285    \@writefile{##1}{\babel@toc{#1}{#2}}}}% %% TODO - ok in plain?
1286 \def\babel@toc#1#2{%
1287  \select@language{#1}}
```

First, check if the user asks for a known language. If so, update the value of \language and call \originalTeX to bring TeX in a certain pre-defined state.

The name of the language is stored in the control sequence \languagename.

Then we have to *re*define \originalTeX to compensate for the things that have been activated. To save memory space for the macro definition of \originalTeX, we construct the control sequence name for the \noextras⟨*lang*⟩ command at definition time by expanding the \csname primitive.

Now activate the language-specific definitions. This is done by constructing the names of three macros by concatenating three words with the argument of \selectlanguage, and calling these macros.

The switching of the values of \lefthyphenmin and \righthyphenmin is somewhat different. First we save their current values, then we check if \⟨*lang*⟩hyphenmins is defined. If it is not, we set default values (2 and 3), otherwise the values in \⟨*lang*⟩hyphenmins will be used.

```
1288 \newif\ifbbl@usedategroup
1289 \def\bbl@switch#1{%  from select@, foreign@
1290  % make sure there is info for the language if so requested
1291  \bbl@ensureinfo{#1}%
1292  % restore
1293  \originalTeX
1294  \expandafter\def\expandafter\originalTeX\expandafter{%
1295    \csname noextras#1\endcsname
1296    \let\originalTeX\@empty
1297    \babel@beginsave}%
1298  \bbl@usehooks{afterreset}{}%
1299  \languageshorthands{none}%
1300  % set the locale id
1301  \bbl@id@assign
1302  % switch captions, date
1303  \ifcase\bbl@select@type
1304    \ifhmode
1305      \hskip\z@skip % trick to ignore spaces
1306      \csname captions#1\endcsname\relax
1307      \csname date#1\endcsname\relax
1308      \loop\ifdim\lastskip>\z@\unskip\repeat\unskip
1309    \else
1310      \csname captions#1\endcsname\relax
1311      \csname date#1\endcsname\relax
1312    \fi
1313  \else
1314    \ifbbl@usedategroup   % if \foreign... within \<lang>date
```

```
1315        \bbl@usedategroupfalse
1316        \ifhmode
1317          \hskip\z@skip % trick to ignore spaces
1318          \csname date#1\endcsname\relax
1319          \loop\ifdim\lastskip>\z@\unskip\repeat\unskip
1320        \else
1321          \csname date#1\endcsname\relax
1322        \fi
1323      \fi
1324    \fi
1325    % switch extras
1326    \bbl@usehooks{beforeextras}{}%
1327    \csname extras#1\endcsname\relax
1328    \bbl@usehooks{afterextras}{}%
1329    %  > babel-ensure
1330    %  > babel-sh-<short>
1331    %  > babel-bidi
1332    %  > babel-fontspec
1333    % hyphenation - case mapping
1334    \ifcase\bbl@opt@hyphenmap\or
1335      \def\BabelLower##1##2{\lccode##1=##2\relax}%
1336      \ifnum\bbl@hymapsel>4\else
1337        \csname\languagename @bbl@hyphenmap\endcsname
1338      \fi
1339      \chardef\bbl@opt@hyphenmap\z@
1340    \else
1341      \ifnum\bbl@hymapsel>\bbl@opt@hyphenmap\else
1342        \csname\languagename @bbl@hyphenmap\endcsname
1343      \fi
1344    \fi
1345    \global\let\bbl@hymapsel\@cclv
1346    % hyphenation - patterns
1347    \bbl@patterns{#1}%
1348    % hyphenation - mins
1349    \babel@savevariable\lefthyphenmin
1350    \babel@savevariable\righthyphenmin
1351    \expandafter\ifx\csname #1hyphenmins\endcsname\relax
1352      \set@hyphenmins\tw@\thr@@\relax
1353    \else
1354      \expandafter\expandafter\expandafter\set@hyphenmins
1355        \csname #1hyphenmins\endcsname\relax
1356    \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.

```
1357 \long\def\otherlanguage#1{%
1358   \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\thr@@\fi
1359   \csname selectlanguage \endcsname{#1}%
1360   \ignorespaces}
```

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

```
1361 \long\def\endotherlanguage{%
1362   \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.

```
1363 \expandafter\def\csname otherlanguage*\endcsname#1{%
1364   \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
1365   \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".

```
1366 \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.

```
1367 \providecommand\bbl@beforeforeign{}
1368 \edef\foreignlanguage{%
1369   \noexpand\protect
1370   \expandafter\noexpand\csname foreignlanguage \endcsname}
1371 \expandafter\def\csname foreignlanguage \endcsname{%
1372   \@ifstar\bbl@foreign@s\bbl@foreign@x}
1373 \def\bbl@foreign@x#1#2{%
1374   \begingroup
1375     \let\BabelText\@firstofone
1376     \bbl@beforeforeign
1377     \foreign@language{#1}%
1378     \bbl@usehooks{foreign}{}%
1379     \BabelText{#2}% Now in horizontal mode!
1380   \endgroup}
1381 \def\bbl@foreign@s#1#2{% TODO - \shapemode, \@setpar, ?\@@par
1382   \begingroup
1383     {\par}%
1384     \let\BabelText\@firstofone
1385     \foreign@language{#1}%
1386     \bbl@usehooks{foreign*}{}%
1387     \bbl@dirparastext
1388     \BabelText{#2}% Still in vertical mode!
```

```
1389        {\par}%
1390     \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.

```
1391 \def\foreign@language#1{%
1392   % set name
1393   \edef\languagename{#1}%
1394   \bbl@fixname\languagename
1395   % TODO. name@map here?
1396   \bbl@provide@locale
1397   \bbl@iflanguage\languagename{%
1398     \expandafter\ifx\csname date\languagename\endcsname\relax
1399       \bbl@warning    % TODO - why a warning, not an error?
1400         {Unknown language `#1'. Either you have\\%
1401          misspelled its name, it has not been installed,\\%
1402          or you requested it in a previous run. Fix its name,\\%
1403          install it or just rerun the file, respectively. In\\%
1404          some cases, you may need to remove the aux file.\\%
1405          I'll proceed, but expect wrong results.\\%
1406          Reported}%
1407     \fi
1408   % set type
1409   \let\bbl@select@type\@ne
1410   \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.

```
1411 \let\bbl@hyphlist\@empty
1412 \let\bbl@hyphenation@\relax
1413 \let\bbl@pttnlist\@empty
1414 \let\bbl@patterns@\relax
1415 \let\bbl@hymapsel=\@cclv
1416 \def\bbl@patterns#1{%
1417   \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
1418       \csname l@#1\endcsname
1419       \edef\bbl@tempa{#1}%
1420     \else
1421       \csname l@#1:\f@encoding\endcsname
1422       \edef\bbl@tempa{#1:\f@encoding}%
1423     \fi
1424   \@expandtwoargs\bbl@usehooks{patterns}{{#1}{\bbl@tempa}}%
1425   % > luatex
1426   \@ifundefined{bbl@hyphenation@}{}{% Can be \relax!
1427     \begingroup
1428     \bbl@xin@{,\number\language,}{,\bbl@hyphlist}%
1429     \ifin@\else
1430       \@expandtwoargs\bbl@usehooks{hyphenation}{{#1}{\bbl@tempa}}%
1431       \hyphenation{%
```

```
1432              \bbl@hyphenation@
1433              \@ifundefined{bbl@hyphenation@#1}%
1434                \@empty
1435                {\space\csname bbl@hyphenation@#1\endcsname}}%
1436            \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
1437         \fi
1438      \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*.

```
1439 \def\hyphenrules#1{%
1440   \edef\bbl@tempf{#1}%
1441   \bbl@fixname\bbl@tempf
1442   \bbl@iflanguage\bbl@tempf{%
1443     \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
1444     \languageshorthands{none}%
1445     \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax
1446       \set@hyphenmins\tw@\thr@@\relax
1447     \else
1448       \expandafter\expandafter\expandafter\set@hyphenmins
1449       \csname\bbl@tempf hyphenmins\endcsname\relax
1450     \fi}}
1451 \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.

```
1452 \def\providehyphenmins#1#2{%
1453   \expandafter\ifx\csname #1hyphenmins\endcsname\relax
1454     \@namedef{#1hyphenmins}{#2}%
1455   \fi}
```

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

```
1456 \def\set@hyphenmins#1#2{%
1457   \lefthyphenmin#1\relax
1458   \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.

```
1459 \ifx\ProvidesFile\@undefined
1460   \def\ProvidesLanguage#1[#2 #3 #4]{%
1461     \wlog{Language: #1 #4 #3 <#2>}%
1462     }
1463 \else
1464   \def\ProvidesLanguage#1{%
1465     \begingroup
1466       \catcode`\ 10 %
1467       \@makeother\/%
1468       \@ifnextchar[%
1469         {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}}
1470   \def\@provideslanguage#1[#2]{%
1471     \wlog{Language: #1 #2}%
```

```
1472        \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
1473      \endgroup}
1474 \fi
```

\originalTeX   The macro\originalTeX should be known to TeX at this moment. As it has to be
expandable we \let it to \@empty instead of \relax.

```
1475 \ifx\originalTeX\@undefined\let\originalTeX\@empty\fi
```

Because this part of the code can be included in a format, we make sure that the macro
which initializes the save mechanism, \babel@beginsave, is not considered to be
undefined.

```
1476 \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':

```
1477 \providecommand\setlocale{%
1478   \bbl@error
1479     {Not yet available}%
1480     {Find an armchair, sit down and wait}}
1481 \let\uselocale\setlocale
1482 \let\locale\setlocale
1483 \let\selectlocale\setlocale
1484 \let\localename\setlocale
1485 \let\textlocale\setlocale
1486 \let\textlanguage\setlocale
1487 \let\languagetext\setlocale
```

9.2   Errors

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

\@noopterr   When the package was loaded without options not everything will work as expected. An
error message is issued in that case.
When the format knows about \PackageError it must be LaTeX 2$_\varepsilon$, so we can safely use its
error handling interface. Otherwise we'll have to 'keep it simple'.
Infos are not written to the console, but on the other hand many people think warnings are
errors, so a further message type is defined: an important info which is sent to the console.

```
1488 \edef\bbl@nulllanguage{\string\language=0}
1489 \ifx\PackageError\@undefined  % TODO. Move to Plain
1490   \def\bbl@error#1#2{%
1491     \begingroup
1492       \newlinechar=`\^^J
1493       \def\\{^^J(babel) }%
1494       \errhelp{#2}\errmessage{\\#1}%
1495     \endgroup}
1496   \def\bbl@warning#1{%
1497     \begingroup
1498       \newlinechar=`\^^J
1499       \def\\{^^J(babel) }%
1500       \message{\\#1}%
1501     \endgroup}
1502   \let\bbl@infowarn\bbl@warning
1503   \def\bbl@info#1{%
```

```
1504      \begingroup
1505        \newlinechar=`\^^J
1506        \def\\{^^J}%
1507        \wlog{#1}%
1508      \endgroup}
1509 \fi
1510 \def\bbl@nocaption{\protect\bbl@nocaption@i}
1511 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
1512    \global\@namedef{#2}{\textbf{?#1?}}%
1513    \@nameuse{#2}%
1514    \bbl@warning{%
1515      \@backslashchar#2 not set. Please, define\\%
1516      it in the preamble with something like:\\%
1517      \string\renewcommand\@backslashchar#2{..}\\%
1518      Reported}}
1519 \def\bbl@tentative{\protect\bbl@tentative@i}
1520 \def\bbl@tentative@i#1{%
1521    \bbl@warning{%
1522      Some functions for '#1' are tentative.\\%
1523      They might not work as expected and their behavior\\%
1524      could change in the future.\\%
1525      Reported}}
1526 \def\@nolanerr#1{%
1527    \bbl@error
1528      {You haven't defined the language #1\space yet.\\%
1529       Perhaps you misspelled it or your installation\\%
1530       is not complete}%
1531      {Your command will be ignored, type <return> to proceed}}
1532 \def\@nopatterns#1{%
1533    \bbl@warning
1534      {No hyphenation patterns were preloaded for\\%
1535       the language `#1' into the format.\\%
1536       Please, configure your TeX system to add them and\\%
1537       rebuild the format. Now I will use the patterns\\%
1538       preloaded for \bbl@nulllanguage\space instead}}
1539 \let\bbl@usehooks\@gobbletwo
1540 \ifx\bbl@onlyswitch\@empty\endinput\fi
1541    % Here ended switch.def
```

 Here ended switch.def.

```
1542 \ifx\directlua\@undefined\else
1543    \ifx\bbl@luapatterns\@undefined
1544      \input luababel.def
1545    \fi
1546 \fi
1547 ⟨⟨Basic macros⟩⟩
1548 \bbl@trace{Compatibility with language.def}
1549 \ifx\bbl@languages\@undefined
1550    \ifx\directlua\@undefined
1551      \openin1 = language.def % TODO. Remove hardcoded number
1552      \ifeof1
1553        \closein1
1554        \message{I couldn't find the file language.def}
1555      \else
1556        \closein1
1557        \begingroup
1558          \def\addlanguage#1#2#3#4#5{%
1559            \expandafter\ifx\csname lang@#1\endcsname\relax\else
1560              \global\expandafter\let\csname l@#1\expandafter\endcsname
```

```
1561              \csname lang@#1\endcsname
1562            \fi}%
1563          \def\uselanguage#1{}%
1564          \input language.def
1565        \endgroup
1566      \fi
1567    \fi
1568    \chardef\l@english\z@
1569 \fi
```

\addto   It takes two arguments, a ⟨*control sequence*⟩ and TEX-code to be added to the ⟨*control sequence*⟩.

If the ⟨*control sequence*⟩ has not been defined before it is defined now. The control sequence could also expand to \relax, in which case a circular definition results. The net result is a stack overflow. Note there is an inconsistency, because the assignment in the last branch is global.

```
1570 \def\addto#1#2{%
1571   \ifx#1\@undefined
1572     \def#1{#2}%
1573   \else
1574     \ifx#1\relax
1575       \def#1{#2}%
1576     \else
1577       {\toks@\expandafter{#1#2}%
1578        \xdef#1{\the\toks@}}%
1579     \fi
1580   \fi}
```

The macro \initiate@active@char below takes all the necessary actions to make its argument a shorthand character. The real work is performed once for each character. But first we define a little tool. TODO. Always used with additional expansions. Move them here? Move the macro to basic?

```
1581 \def\bbl@withactive#1#2{%
1582   \begingroup
1583     \lccode`~=`#2\relax
1584     \lowercase{\endgroup#1~}}
```

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

```
1585 \def\bbl@redefine#1{%
1586   \edef\bbl@tempa{\bbl@stripslash#1}%
1587   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1588   \expandafter\def\csname\bbl@tempa\endcsname}
1589 \@onlypreamble\bbl@redefine
```

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

```
1590 \def\bbl@redefine@long#1{%
1591   \edef\bbl@tempa{\bbl@stripslash#1}%
1592   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1593   \expandafter\long\expandafter\def\csname\bbl@tempa\endcsname}
1594 \@onlypreamble\bbl@redefine@long
```

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

```
1595 \def\bbl@redefinerobust#1{%
1596   \edef\bbl@tempa{\bbl@stripslash#1}%
1597   \bbl@ifunset{\bbl@tempa\space}%
1598     {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
1599      \bbl@exp{\def\\#1{\\\protect\<\bbl@tempa\space>}}}%
1600     {\bbl@exp{\let\<org@\bbl@tempa>\<\bbl@tempa\space>}}%
1601   \@namedef{\bbl@tempa\space}}
1602 \@onlypreamble\bbl@redefinerobust
```

9.3 Hooks

Admittedly, the current implementation is a somewhat simplistic and does very little to catch errors, but it is meant for developers, after all. \bbl@usehooks is the commands used by babel to execute hooks defined for an event.

```
1603 \bbl@trace{Hooks}
1604 \newcommand\AddBabelHook[3][]{%
1605   \bbl@ifunset{bbl@hk@#2}{\EnableBabelHook{#2}}{}%
1606   \def\bbl@tempa##1,#3=##2,##3\@empty{\def\bbl@tempb{##2}}%
1607   \expandafter\bbl@tempa\bbl@evargs,#3=,\@empty
1608   \bbl@ifunset{bbl@ev@#2@#3@#1}%
1609     {\bbl@csarg\bbl@add{ev@#3@#1}{\bbl@elt{#2}}}%
1610     {\bbl@csarg\let{ev@#2@#3@#1}\relax}%
1611   \bbl@csarg\newcommand{ev@#2@#3@#1}[\bbl@tempb]}
1612 \newcommand\EnableBabelHook[1]{\bbl@csarg\let{hk@#1}\@firstofone}
1613 \newcommand\DisableBabelHook[1]{\bbl@csarg\let{hk@#1}\@gobble}
1614 \def\bbl@usehooks#1#2{%
1615   \def\bbl@elt##1{%
1616     \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#1@}#2}}%
1617   \bbl@cs{ev@#1@}%
1618   \ifx\languagename\@undefined\else % Test required for Plain (?)
1619     \def\bbl@elt##1{%
1620       \bbl@cs{hk@##1}{\bbl@cl{ev@##1@#1}#2}}%
1621     \bbl@cl{ev@#1}%
1622   \fi}
```

To ensure forward compatibility, arguments in hooks are set implicitly. So, if a further argument is added in the future, there is no need to change the existing code. Note events intended for hyphen.cfg are also loaded (just in case you need them for some reason).

```
1623 \def\bbl@evargs{,% <- don't delete this comma
1624   everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
1625   adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
1626   beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
1627   hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%
1628   beforestart=0,languagename=2}
```

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

The macro \bbl@e@⟨*language*⟩ contains \bbl@ensure{⟨*include*⟩}{⟨*exclude*⟩}{⟨*fontenc*⟩}, which in in turn loops over the macros names in \bbl@captionslist, excluding (with the help of \in@) those in the exclude list. If the fontenc is given (and not \relax), the

\fontencoding is also added. Then we loop over the include list, but if the macro already contains \foreignlanguage, nothing is done. Note this macro (1) is not restricted to the preamble, and (2) changes are local.

```
1629 \bbl@trace{Defining babelensure}
1630 \newcommand\babelensure[2][]{%  TODO - revise test files
1631   \AddBabelHook{babel-ensure}{afterextras}{%
1632     \ifcase\bbl@select@type
1633       \bbl@cl{e}%
1634     \fi}%
1635   \begingroup
1636     \let\bbl@ens@include\@empty
1637     \let\bbl@ens@exclude\@empty
1638     \def\bbl@ens@fontenc{\relax}%
1639     \def\bbl@tempb##1{%
1640       \ifx\@empty##1\else\noexpand##1\expandafter\bbl@tempb\fi}%
1641     \edef\bbl@tempa{\bbl@tempb#1\@empty}%
1642     \def\bbl@tempb##1=##2\@@{\@namedef{bbl@ens@##1}{##2}}%
1643     \bbl@foreach\bbl@tempa{\bbl@tempb##1\@@}%
1644     \def\bbl@tempc{\bbl@ensure}%
1645     \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1646       \expandafter{\bbl@ens@include}}%
1647     \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1648       \expandafter{\bbl@ens@exclude}}%
1649     \toks@\expandafter{\bbl@tempc}%
1650     \bbl@exp{%
1651   \endgroup
1652   \def\<bbl@e@#2>{\the\toks@{\bbl@ens@fontenc}}}}}
1653 \def\bbl@ensure#1#2#3{% 1: include 2: exclude 3: fontenc
1654   \def\bbl@tempb##1{% elt for (excluding) \bbl@captionslist list
1655     \ifx##1\@undefined % 3.32 - Don't assume the macro exists
1656       \edef##1{\noexpand\bbl@nocaption
1657         {\bbl@stripslash##1}{\languagename\bbl@stripslash##1}}%
1658     \fi
1659     \ifx##1\@empty\else
1660       \in@{##1}{#2}%
1661       \ifin@\else
1662         \bbl@ifunset{bbl@ensure@\languagename}%
1663           {\bbl@exp{%
1664             \\\DeclareRobustCommand\<bbl@ensure@\languagename>[1]{%
1665               \\\foreignlanguage{\languagename}%
1666               {\ifx\relax#3\else
1667                 \\\fontencoding{#3}\\\selectfont
1668               \fi
1669               ########1}}}}%
1670           {}%
1671         \toks@\expandafter{##1}%
1672         \edef##1{%
1673           \bbl@csarg\noexpand{ensure@\languagename}%
1674           {\the\toks@}}%
1675       \fi
1676       \expandafter\bbl@tempb
1677     \fi}%
1678   \expandafter\bbl@tempb\bbl@captionslist\today\@empty
1679   \def\bbl@tempa##1{% elt for include list
1680     \ifx##1\@empty\else
1681       \bbl@csarg\in@{ensure@\languagename\expandafter}\expandafter{##1}%
1682       \ifin@\else
1683         \bbl@tempb##1\@empty
```

99

```
1684        \fi
1685        \expandafter\bbl@tempa
1686      \fi}%
1687   \bbl@tempa#1\@empty}
1688 \def\bbl@captionslist{%
1689   \prefacename\refname\abstractname\bibname\chaptername\appendixname
1690   \contentsname\listfigurename\listtablename\indexname\figurename
1691   \tablename\partname\enclname\ccname\headtoname\pagename\seename
1692   \alsoname\proofname\glossaryname}
```

9.4 Setting up language files

\LdfInit   \LdfInit macro takes two arguments. The first argument is the name of the language that
will be defined in the language definition file; the second argument is either a control
sequence or a string from which a control sequence should be constructed. The existence
of the control sequence indicates that the file has been processed before.

At the start of processing a language definition file we always check the category code of
the at-sign. We make sure that it is a 'letter' during the processing of the file. We also save
its name as the last called option, even if not loaded.

Another character that needs to have the correct category code during processing of
language definition files is the equals sign, '=', because it is sometimes used in constructions
with the \let primitive. Therefore we store its current catcode and restore it later on.

Now we check whether we should perhaps stop the processing of this file. To do this we
first need to check whether the second argument that is passed to \LdfInit is a control
sequence. We do that by looking at the first token after passing #2 through string. When
it is equal to \@backslashchar we are dealing with a control sequence which we can
compare with \@undefined.

If so, we call \ldf@quit to set the main language, restore the category code of the @-sign
and call \endinput

When #2 was *not* a control sequence we construct one and compare it with \relax.
Finally we check \originalTeX.

```
1693 \bbl@trace{Macros for setting language files up}
1694 \def\bbl@ldfinit{% TODO. Merge into the next macro? Unused elsewhere
1695   \let\bbl@screset\@empty
1696   \let\BabelStrings\bbl@opt@string
1697   \let\BabelOptions\@empty
1698   \let\BabelLanguages\relax
1699   \ifx\originalTeX\@undefined
1700     \let\originalTeX\@empty
1701   \else
1702     \originalTeX
1703   \fi}
1704 \def\LdfInit#1#2{%
1705   \chardef\atcatcode=\catcode`\@
1706   \catcode`\@=11\relax
1707   \chardef\eqcatcode=\catcode`\=
1708   \catcode`\==12\relax
1709   \expandafter\if\expandafter\@backslashchar
1710                  \expandafter\@car\string#2\@nil
1711     \ifx#2\@undefined\else
1712       \ldf@quit{#1}%
1713     \fi
1714   \else
1715     \expandafter\ifx\csname#2\endcsname\relax\else
1716       \ldf@quit{#1}%
1717     \fi
```

1718 \fi
                          1719 \bbl@ldfinit}

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

```
1720 \def\ldf@quit#1{%
1721   \expandafter\main@language\expandafter{#1}%
1722   \catcode`\@=\atcatcode \let\atcatcode\relax
1723   \catcode`\==\eqcatcode \let\eqcatcode\relax
1724   \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.

```
1725 \def\bbl@afterldf#1{% TODO. Merge into the next macro? Unused elsewhere
1726   \bbl@afterlang
1727   \let\bbl@afterlang\relax
1728   \let\BabelModifiers\relax
1729   \let\bbl@screset\relax}%
1730 \def\ldf@finish#1{%
1731   \ifx\loadlocalcfg\@undefined\else % For LaTeX 209
1732     \loadlocalcfg{#1}%
1733   \fi
1734   \bbl@afterldf{#1}%
1735   \expandafter\main@language\expandafter{#1}%
1736   \catcode`\@=\atcatcode \let\atcatcode\relax
1737   \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.

```
1738 \@onlypreamble\LdfInit
1739 \@onlypreamble\ldf@quit
1740 \@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.

```
1741 \def\main@language#1{%
1742   \def\bbl@main@language{#1}%
1743   \let\languagename\bbl@main@language % TODO. Set localename
1744   \bbl@id@assign
1745   \bbl@patterns{\languagename}}
```

We also have to make sure that some code gets executed at the beginning of the document,
either when the aux file is read or, if it does not exist, when the \AtBeginDocument is
executed. Languages do not set \pagedir, so we set here for the whole document to the
main \bodydir.

```
1746 \def\bbl@beforestart{%
1747   \bbl@usehooks{beforestart}{}%
1748   \global\let\bbl@beforestart\relax}
1749 \AtBeginDocument{%
1750   \@nameuse{bbl@beforestart}%
1751   \if@filesw
1752     \immediate\write\@mainaux{\string\@nameuse{bbl@beforestart}}%
1753   \fi
1754   \expandafter\selectlanguage\expandafter{\bbl@main@language}%
```

```
1755    \ifbbl@single  % must go after the line above.
1756      \renewcommand\selectlanguage[1]{}%
1757      \renewcommand\foreignlanguage[2]{#2}%
1758      \global\let\babel@aux\@gobbletwo  % Also as flag
1759    \fi
1760    \ifcase\bbl@engine\or\pagedir\bodydir\fi} % TODO - a better place
```

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

```
1761 \def\select@language@x#1{%
1762    \ifcase\bbl@select@type
1763      \bbl@ifsamestring\languagename{#1}{}{\select@language{#1}}%
1764    \else
1765      \select@language{#1}%
1766    \fi}
```

9.5   Shorthands

\bbl@add@special   The macro \bbl@add@special is used to add a new character (or single character control sequence) to the macro \dospecials (and \@sanitize if LaTeX is used). It is used only at one place, namely when \initiate@active@char is called (which is ignored if the char has been made active before). Because \@sanitize can be undefined, we put the definition inside a conditional.
Items are added to the lists without checking its existence or the original catcode. It does not hurt, but should be fixed. It's already done with \nfss@catcodes, added in 3.10.

```
1767 \bbl@trace{Shorhands}
1768 \def\bbl@add@special#1{% 1:a macro like \", \?, etc.
1769    \bbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
1770    \bbl@ifunset{@sanitize}{}{\bbl@add\@sanitize{\@makeother#1}}%
1771    \ifx\nfss@catcodes\@undefined\else % TODO - same for above
1772      \begingroup
1773        \catcode`#1\active
1774        \nfss@catcodes
1775        \ifnum\catcode`#1=\active
1776          \endgroup
1777          \bbl@add\nfss@catcodes{\@makeother#1}%
1778        \else
1779          \endgroup
1780        \fi
1781    \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.

```
1782 \def\bbl@remove@special#1{%
1783    \begingroup
1784      \def\x##1##2{\ifnum`#1=`##2\noexpand\@empty
1785                  \else\noexpand##1\noexpand##2\fi}%
1786      \def\do{\x\do}%
1787      \def\@makeother{\x\@makeother}%
1788    \edef\x{\endgroup
1789      \def\noexpand\dospecials{\dospecials}%
1790      \expandafter\ifx\csname @sanitize\endcsname\relax\else
1791        \def\noexpand\@sanitize{\@sanitize}%
1792      \fi}%
1793    \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

102

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).

```
1794 \def\bbl@active@def#1#2#3#4{%
1795   \@namedef{#3#1}{%
1796     \expandafter\ifx\csname#2@sh@#1@\endcsname\relax
1797       \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
1798     \else
1799       \bbl@afterfi\csname#2@sh@#1@\endcsname
1800     \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.

```
1801   \long\@namedef{#3@arg#1}##1{%
1802     \expandafter\ifx\csname#2@sh@#1@\string##1@\endcsname\relax
1803       \bbl@afterelse\csname#4#1\endcsname##1%
1804     \else
1805       \bbl@afterfi\csname#2@sh@#1@\string##1@\endcsname
1806     \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.

```
1807 \def\initiate@active@char#1{%
1808   \bbl@ifunset{active@char\string#1}%
1809     {\bbl@withactive
1810       {\expandafter\@initiate@active@char\expandafter}#1\string#1#1}%
1811     {}}
```

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).

```
1812 \def\@initiate@active@char#1#2#3{%
1813   \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
1814   \ifx#1\@undefined
1815     \bbl@csarg\edef{oridef@#2}{\let\noexpand#1\noexpand\@undefined}%
1816   \else
1817     \bbl@csarg\let{oridef@@#2}#1%
1818     \bbl@csarg\edef{oridef@#2}{%
1819       \let\noexpand#1%
1820       \expandafter\noexpand\csname bbl@oridef@@#2\endcsname}%
1821   \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*).

```
1822  \ifx#1#3\relax
1823    \expandafter\let\csname normal@char#2\endcsname#3%
1824  \else
1825    \bbl@info{Making #2 an active character}%
1826    \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
1827      \@namedef{normal@char#2}{%
1828        \textormath{#3}{\csname bbl@oridef@@#2\endcsname}}%
1829    \else
1830      \@namedef{normal@char#2}{#3}%
1831    \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).

```
1832    \bbl@restoreactive{#2}%
1833    \AtBeginDocument{%
1834      \catcode`#2\active
1835      \if@filesw
1836        \immediate\write\@mainaux{\catcode`\string#2\active}%
1837      \fi}%
1838    \expandafter\bbl@add@special\csname#2\endcsname
1839    \catcode`#2\active
1840  \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*⟩).

```
1841  \let\bbl@tempa\@firstoftwo
1842  \if\string^#2%
1843    \def\bbl@tempa{\noexpand\textormath}%
1844  \else
1845    \ifx\bbl@mathnormal\@undefined\else
1846      \let\bbl@tempa\bbl@mathnormal
1847    \fi
1848  \fi
1849  \expandafter\edef\csname active@char#2\endcsname{%
1850    \bbl@tempa
1851      {\noexpand\if@safe@actives
1852        \noexpand\expandafter
1853        \expandafter\noexpand\csname normal@char#2\endcsname
1854      \noexpand\else
1855        \noexpand\expandafter
1856        \expandafter\noexpand\csname bbl@doactive#2\endcsname
1857      \noexpand\fi}%
1858    {\expandafter\noexpand\csname normal@char#2\endcsname}}%
```

```
1859   \bbl@csarg\edef{doactive#2}{%
1860     \expandafter\noexpand\csname user@active#2\endcsname}%
```

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

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

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

```
1861   \bbl@csarg\edef{active@#2}{%
1862     \noexpand\active@prefix\noexpand#1%
1863     \expandafter\noexpand\csname active@char#2\endcsname}%
1864   \bbl@csarg\edef{normal@#2}{%
1865     \noexpand\active@prefix\noexpand#1%
1866     \expandafter\noexpand\csname normal@char#2\endcsname}%
1867   \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.

```
1868   \bbl@active@def#2\user@group{user@active}{language@active}%
1869   \bbl@active@def#2\language@group{language@active}{system@active}%
1870   \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.

```
1871   \expandafter\edef\csname\user@group @sh@#2@@\endcsname
1872     {\expandafter\noexpand\csname normal@char#2\endcsname}%
1873   \expandafter\edef\csname\user@group @sh@#2@\string\protect@\endcsname
1874     {\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.

```
1875   \if\string'#2%
1876     \let\prim@s\bbl@prim@s
1877     \let\active@math@prime#1%
1878   \fi
1879   \bbl@usehooks{initiateactive}{{#1}{#2}{#3}}}}
```

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

```
1880 ⟨⟨∗More package options⟩⟩ ≡
1881 \DeclareOption{math=active}{}
1882 \DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
1883 ⟨⟨/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.

```
1884 \@ifpackagewith{babel}{KeepShorthandsActive}%
1885   {\let\bbl@restoreactive\@gobble}%
1886   {\def\bbl@restoreactive#1{%
1887     \bbl@exp{%
```

```
1888        \\\AfterBabelLanguage\\\CurrentOption
1889          {\catcode`#1=\the\catcode`#1\relax}%
1890        \\\AtEndOfPackage
1891          {\catcode`#1=\the\catcode`#1\relax}}}%
1892    \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.

```
1893 \def\bbl@sh@select#1#2{%
1894   \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
1895     \bbl@afterelse\bbl@scndcs
1896   \else
1897     \bbl@afterfi\csname#1@sh@#2@sel\endcsname
1898   \fi}
```

\active@prefix   The command \active@prefix which is used in the expansion of active characters has a
function similar to \OT1-cmd in that it \protects the active character whenever \protect
is *not* \@typeset@protect. The \@gobble is needed to remove a token such as
\activechar: (when the double colon was the active character to be dealt with). There are
two definitions, depending of \ifincsname is available. If there is, the expansion will be
more robust.

```
1899 \begingroup
1900 \bbl@ifunset{ifincsname}% TODO. Ugly. Correct?
1901   {\gdef\active@prefix#1{%
1902      \ifx\protect\@typeset@protect
1903      \else
1904        \ifx\protect\@unexpandable@protect
1905          \noexpand#1%
1906        \else
1907          \protect#1%
1908        \fi
1909        \expandafter\@gobble
1910      \fi}}
1911   {\gdef\active@prefix#1{%
1912      \ifincsname
1913        \string#1%
1914        \expandafter\@gobble
1915      \else
1916        \ifx\protect\@typeset@protect
1917        \else
1918          \ifx\protect\@unexpandable@protect
1919            \noexpand#1%
1920          \else
1921            \protect#1%
1922          \fi
1923          \expandafter\expandafter\expandafter\@gobble
1924        \fi
1925      \fi}}
1926 \endgroup
```

\if@safe@actives   In some circumstances it is necessary to be able to change the expansion of an active
character on the fly. For this purpose the switch @safe@actives is available. The setting of
this switch should be checked in the first level expansion of \active@char⟨char⟩.

```
1927 \newif\if@safe@actives
1928 \@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.

```
1929 \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.

```
1930 \def\bbl@activate#1{%
1931   \bbl@withactive{\expandafter\let\expandafter}#1%
1932     \csname bbl@active@\string#1\endcsname}
1933 \def\bbl@deactivate#1{%
1934   \bbl@withactive{\expandafter\let\expandafter}#1%
1935     \csname bbl@normal@\string#1\endcsname}
```

\bbl@firstcs These macros are used only as a trick when declaring shorthands.
\bbl@scndcs
```
1936 \def\bbl@firstcs#1#2{\csname#1\endcsname}
1937 \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.

```
1938 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
1939 \def\@decl@short#1#2#3\@nil#4{%
1940   \def\bbl@tempa{#3}%
1941   \ifx\bbl@tempa\@empty
1942     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
1943     \bbl@ifunset{#1@sh@\string#2@}{}%
1944       {\def\bbl@tempa{#4}%
1945        \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbl@tempa
1946        \else
1947          \bbl@info
1948            {Redefining #1 shorthand \string#2\\%
1949             in language \CurrentOption}%
1950        \fi}%
1951     \@namedef{#1@sh@\string#2@}{#4}%
1952   \else
1953     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs
1954     \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%
1955       {\def\bbl@tempa{#4}%
1956        \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
1957        \else
1958          \bbl@info
1959            {Redefining #1 shorthand \string#2\string#3\\%
1960             in language \CurrentOption}%
1961        \fi}%
1962     \@namedef{#1@sh@\string#2@\string#3@}{#4}%
1963   \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.

```
1964 \def\textormath{%
1965   \ifmmode
1966     \expandafter\@secondoftwo
1967   \else
1968     \expandafter\@firstoftwo
1969   \fi}
```

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

```
1970 \def\user@group{user}
1971 \def\language@group{english} % TODO. I don't like defaults
1972 \def\system@group{system}
```

\useshorthands This is the user level macro. It initializes and activates the character for use as a shorthand character (ie, it's active in the preamble). Languages can deactivate shorthands, so a starred version is also provided which activates them always after the language has been switched.

```
1973 \def\useshorthands{%
1974   \@ifstar\bbl@usesh@s{\bbl@usesh@x{}}}
1975 \def\bbl@usesh@s#1{%
1976   \bbl@usesh@x
1977     {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bbl@activate{#1}}}%
1978     {#1}}
1979 \def\bbl@usesh@x#1#2{%
1980   \bbl@ifshorthand{#2}%
1981     {\def\user@group{user}%
1982      \initiate@active@char{#2}%
1983      #1%
1984      \bbl@activate{#2}}%
1985     {\bbl@error
1986       {Cannot declare a shorthand turned off (\string#2)}
1987       {Sorry, but you cannot use shorthands which have been\\%
1988        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.

```
1989 \def\user@language@group{user@\language@group}
1990 \def\bbl@set@user@generic#1#2{%
1991   \bbl@ifunset{user@generic@active#1}%
1992     {\bbl@active@def#1\user@language@group{user@active}{user@generic@active}%
1993      \bbl@active@def#1\user@group{user@generic@active}{language@active}%
1994      \expandafter\edef\csname#2@sh@#1@@\endcsname{%
1995        \expandafter\noexpand\csname normal@char#1\endcsname}%
1996      \expandafter\edef\csname#2@sh@#1@\string\protect@\endcsname{%
1997        \expandafter\noexpand\csname user@active#1\endcsname}}%
1998   \@empty}
1999 \newcommand\defineshorthand[3][user]{%
2000   \edef\bbl@tempa{\zap@space#1 \@empty}%
2001   \bbl@for\bbl@tempb\bbl@tempa{%
2002     \if*\expandafter\@car\bbl@tempb\@nil
```

```
2003      \edef\bbl@tempb{user@\expandafter\@gobble\bbl@tempb}%
2004      \@expandtwoargs
2005        \bbl@set@user@generic{\expandafter\string\@car#2\@nil}\bbl@tempb
2006    \fi
2007    \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 [TODO. Unclear].

```
2008 \def\languageshorthands#1{\def\language@group{#1}}
```

\aliasshorthand First the new shorthand needs to be initialized. Then, we define the new shorthand in terms of the original one, but note with \aliasshorthands{"}{/} is \active@prefix /\active@char/, so we still need to let the lattest to \active@char".

```
2009 \def\aliasshorthand#1#2{%
2010   \bbl@ifshorthand{#2}%
2011     {\expandafter\ifx\csname active@char\string#2\endcsname\relax
2012        \ifx\document\@notprerr
2013          \@notshorthand{#2}%
2014        \else
2015          \initiate@active@char{#2}%
2016          \expandafter\let\csname active@char\string#2\expandafter\endcsname
2017            \csname active@char\string#1\endcsname
2018          \expandafter\let\csname normal@char\string#2\expandafter\endcsname
2019            \csname normal@char\string#1\endcsname
2020          \bbl@activate{#2}%
2021        \fi
2022      \fi}%
2023     {\bbl@error
2024        {Cannot declare a shorthand turned off (\string#2)}
2025        {Sorry, but you cannot use shorthands which have been\\%
2026         turned off in the package options}}}
```

\@notshorthand

```
2027 \def\@notshorthand#1{%
2028   \bbl@error{%
2029     The character `\string #1' should be made a shorthand character;\\%
2030     add the command \string\useshorthands\string{#1\string} to
2031     the preamble.\\%
2032     I will ignore your instruction}%
2033    {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.

```
2034 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
2035 \DeclareRobustCommand*\shorthandoff{%
2036   \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
2037 \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.

```
2038 \def\bbl@switch@sh#1#2{%
2039   \ifx#2\@nnil\else
2040     \bbl@ifunset{bbl@active@\string#2}%
2041       {\bbl@error
2042         {I cannot switch `\string#2' on or off--not a shorthand}%
2043         {This character is not a shorthand. Maybe you made\\%
2044          a typing mistake? I will ignore your instruction}}%
2045       {\ifcase#1%
2046         \catcode`#212\relax
2047       \or
2048         \catcode`#2\active
2049       \or
2050         \csname bbl@oricat@\string#2\endcsname
2051         \csname bbl@oridef@\string#2\endcsname
2052       \fi}%
2053     \bbl@afterfi\bbl@switch@sh#1%
2054   \fi}
```

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

```
2055 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
2056 \def\bbl@putsh#1{%
2057   \bbl@ifunset{bbl@active@\string#1}%
2058     {\bbl@putsh@i#1\@empty\@nnil}%
2059     {\csname bbl@active@\string#1\endcsname}}
2060 \def\bbl@putsh@i#1#2\@nnil{%
2061   \csname\languagename @sh@\string#1@%
2062     \ifx\@empty#2\else\string#2@\fi\endcsname}
2063 \ifx\bbl@opt@shorthands\@nnil\else
2064   \let\bbl@s@initiate@active@char\initiate@active@char
2065   \def\initiate@active@char#1{%
2066     \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
2067   \let\bbl@s@switch@sh\bbl@switch@sh
2068   \def\bbl@switch@sh#1#2{%
2069     \ifx#2\@nnil\else
2070       \bbl@afterfi
2071       \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
2072     \fi}
2073   \let\bbl@s@activate\bbl@activate
2074   \def\bbl@activate#1{%
2075     \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
2076   \let\bbl@s@deactivate\bbl@deactivate
2077   \def\bbl@deactivate#1{%
2078     \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
2079 \fi
```

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

```
2080 \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.

```
2081 \def\bbl@prim@s{%
2082   \prime\futurelet\@let@token\bbl@pr@m@s}
2083 \def\bbl@if@primes#1#2{%
2084   \ifx#1\@let@token
2085     \expandafter\@firstoftwo
2086   \else\ifx#2\@let@token
2087     \bbl@afterelse\expandafter\@firstoftwo
2088   \else
2089     \bbl@afterfi\expandafter\@secondoftwo
2090   \fi\fi}
2091 \begingroup
2092   \catcode`\^=7  \catcode`\*=\active  \lccode`\*=`\^
2093   \catcode`\'=12 \catcode`\"=\active  \lccode`\"=`\'
2094   \lowercase{%
2095     \gdef\bbl@pr@m@s{%
2096       \bbl@if@primes"'%
2097         \pr@@@s
2098         {\bbl@if@primes*^\pr@@@t\egroup}}}
2099 \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).

```
2100 \initiate@active@char{~}
2101 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
2102 \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.

```
2103 \expandafter\def\csname OT1dqpos\endcsname{127}
2104 \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

```
2105 \ifx\f@encoding\@undefined
2106   \def\f@encoding{OT1}
2107 \fi
```

9.6   Language attributes

Language attributes provide a means to give the user control over which features of the language definition files he wants to enable.

\languageattribute The macro \languageattribute checks whether its arguments are valid and then activates the selected language attribute. First check whether the language is known, and then process each attribute in the list.

```
2108 \bbl@trace{Language attributes}
2109 \newcommand\languageattribute[2]{%
2110   \def\bbl@tempc{#1}%
2111   \bbl@fixname\bbl@tempc
2112   \bbl@iflanguage\bbl@tempc{%
2113     \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.

```
2114        \ifx\bbl@known@attribs\@undefined
2115          \in@false
2116        \else
2117          \bbl@xin@{,\bbl@tempc-##1,}{,\bbl@known@attribs,}%
2118        \fi
2119        \ifin@
2120          \bbl@warning{%
2121            You have more than once selected the attribute '##1'\\%
2122            for language #1. Reported}%
2123        \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.

```
2124        \bbl@exp{%
2125          \\\bbl@add@list\\\bbl@known@attribs{\bbl@tempc-##1}}%
2126        \edef\bbl@tempa{\bbl@tempc-##1}%
2127        \expandafter\bbl@ifknown@ttrib\expandafter{\bbl@tempa}\bbl@attributes%
2128        {\csname\bbl@tempc @attr@##1\endcsname}%
2129        {\@attrerr{\bbl@tempc}{##1}}%
2130      \fi}}}
2131 \@onlypreamble\languageattribute
```

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

```
2132 \newcommand*{\@attrerr}[2]{%
2133   \bbl@error
2134     {The attribute #2 is unknown for language #1.}%
2135     {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}.

```
2136 \def\bbl@declare@ttribute#1#2#3{%
2137   \bbl@xin@{,#2,}{,\BabelModifiers,}%
2138   \ifin@
2139     \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
2140   \fi
2141   \bbl@add@list\bbl@attributes{#1-#2}%
2142   \expandafter\def\csname#1@attr@#2\endcsname{#3}}
```

\bbl@ifattributeset   This internal macro has 4 arguments. It can be used to interpret TeX code based on whether a certain attribute was set. This command should appear inside the argument to \AtBeginDocument because the attributes are set in the document preamble, *after* babel is loaded.

The first argument is the language, the second argument the attribute being checked, and the third and fourth arguments are the true and false clauses.

First we need to find out if any attributes were set; if not we're done. Then we need to check the list of known attributes. When we're this far \ifin@ has a value indicating if the attribute in question was set or not. Just to be safe the code to be executed is 'thrown over the \fi'.

```
2143 \def\bbl@ifattributeset#1#2#3#4{%
```

```
2144    \ifx\bbl@known@attribs\@undefined
2145      \in@false
2146    \else
2147      \bbl@xin@{,#1-#2,}{,\bbl@known@attribs,}%
2148    \fi
2149    \ifin@
2150      \bbl@afterelse#3%
2151    \else
2152      \bbl@afterfi#4%
2153    \fi
2154    }
```

\bbl@ifknown@ttrib   An internal macro to check whether a given language/attribute is known. The macro takes
4 arguments, the language/attribute, the attribute list, the TeX-code to be executed when
the attribute is known and the TeX-code to be executed otherwise.
We first assume the attribute is unknown. Then we loop over the list of known attributes,
trying to find a match. When a match is found the definition of \bbl@tempa is changed.
Finally we execute \bbl@tempa.

```
2155 \def\bbl@ifknown@ttrib#1#2{%
2156    \let\bbl@tempa\@secondoftwo
2157    \bbl@loopx\bbl@tempb{#2}{%
2158      \expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{,#1,}%
2159      \ifin@
2160        \let\bbl@tempa\@firstoftwo
2161      \else
2162      \fi}%
2163    \bbl@tempa
2164 }
```

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

```
2165 \def\bbl@clear@ttribs{%
2166    \ifx\bbl@attributes\@undefined\else
2167      \bbl@loopx\bbl@tempa{\bbl@attributes}{%
2168        \expandafter\bbl@clear@ttrib\bbl@tempa.
2169        }%
2170      \let\bbl@attributes\@undefined
2171    \fi}
2172 \def\bbl@clear@ttrib#1-#2.{%
2173    \expandafter\let\csname#1@attr@#2\endcsname\@undefined}
2174 \AtBeginDocument{\bbl@clear@ttribs}
```

9.7   Support for saving macro definitions

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

\babel@savecnt   The initialization of a new save cycle: reset the counter to zero.
\babel@beginsave
```
2175 \bbl@trace{Macros for saving definitions}
2176 \def\babel@beginsave{\babel@savecnt\z@}
```

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

```
2177 \newcount\babel@savecnt
2178 \babel@beginsave
```

\babel@save
\babel@savevariable
The macro \babel@save⟨*csname*⟩ saves the current meaning of the control sequence ⟨*csname*⟩ to \originalTeX[31]. To do this, we let the current meaning to a temporary control sequence, the restore commands are appended to \originalTeX and the counter is incremented. The macro \babel@savevariable⟨*variable*⟩ saves the value of the variable. ⟨*variable*⟩ can be anything allowed after the \the primitive.

```
2179 \def\babel@save#1{%
2180   \expandafter\let\csname babel@\number\babel@savecnt\endcsname#1\relax
2181   \toks@\expandafter{\originalTeX\let#1=}%
2182   \bbl@exp{%
2183     \def\\\originalTeX{\the\toks@\<babel@\number\babel@savecnt>\relax}}%
2184   \advance\babel@savecnt\@ne}
2185 \def\babel@savevariable#1{%
2186   \toks@\expandafter{\originalTeX #1=}%
2187   \bbl@exp{\def\\\originalTeX{\the\toks@\the#1\relax}}}
```

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

```
2188 \def\bbl@frenchspacing{%
2189   \ifnum\the\sfcode`\.=\@m
2190     \let\bbl@nonfrenchspacing\relax
2191   \else
2192     \frenchspacing
2193     \let\bbl@nonfrenchspacing\nonfrenchspacing
2194   \fi}
2195 \let\bbl@nonfrenchspacing\nonfrenchspacing
```

9.8 Short tags

\babeltags
This macro is straightforward. After zapping spaces, we loop over the list and define the macros \text⟨*tag*⟩ and \⟨*tag*⟩. Definitions are first expanded so that they don't contain \csname but the actual macro.

```
2196 \bbl@trace{Short tags}
2197 \def\babeltags#1{%
2198   \edef\bbl@tempa{\zap@space#1 \@empty}%
2199   \def\bbl@tempb##1=##2\@@{%
2200     \edef\bbl@tempc{%
2201       \noexpand\newcommand
2202       \expandafter\noexpand\csname ##1\endcsname{%
2203         \noexpand\protect
2204         \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}%
2205       \noexpand\newcommand
2206       \expandafter\noexpand\csname text##1\endcsname{%
2207         \noexpand\foreignlanguage{##2}}}%
2208     \bbl@tempc}%
2209   \bbl@for\bbl@tempa\bbl@tempa{%
2210     \expandafter\bbl@tempb\bbl@tempa\@@}}
```

---

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

9.9 Hyphens

\babelhyphenation   This macro saves hyphenation exceptions. Two macros are used to store them: \bbl@hyphenation@ for the global ones and \bbl@hyphenation<lang> for language ones. See \bbl@patterns above for further details. We make sure there is a space between words when multiple commands are used.

```
2211 \bbl@trace{Hyphens}
2212 \@onlypreamble\babelhyphenation
2213 \AtEndOfPackage{%
2214   \newcommand\babelhyphenation[2][\@empty]{%
2215     \ifx\bbl@hyphenation@\relax
2216       \let\bbl@hyphenation@\@empty
2217     \fi
2218     \ifx\bbl@hyphlist\@empty\else
2219       \bbl@warning{%
2220         You must not intermingle \string\selectlanguage\space and\\%
2221         \string\babelhyphenation\space or some exceptions will not\\%
2222         be taken into account. Reported}%
2223     \fi
2224     \ifx\@empty#1%
2225       \protected@edef\bbl@hyphenation@{\bbl@hyphenation@\space#2}%
2226     \else
2227       \bbl@vforeach{#1}{%
2228         \def\bbl@tempa{##1}%
2229         \bbl@fixname\bbl@tempa
2230         \bbl@iflanguage\bbl@tempa{%
2231           \bbl@csarg\protected@edef{hyphenation@\bbl@tempa}{%
2232             \bbl@ifunset{bbl@hyphenation@\bbl@tempa}%
2233               \@empty
2234               {\csname bbl@hyphenation@\bbl@tempa\endcsname\space}%
2235             #2}}%
2236     \fi}}
```

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

```
2237 \def\bbl@allowhyphens{\ifvmode\else\nobreak\hskip\z@skip\fi}
2238 \def\bbl@t@one{T1}
2239 \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.

```
2240 \newcommand\babelnullhyphen{\char\hyphenchar\font}
2241 \def\babelhyphen{\active@prefix\babelhyphen\bbl@hyphen}
2242 \def\bbl@hyphen{%
2243   \@ifstar{\bbl@hyphen@i @}{\bbl@hyphen@i\@empty}}
2244 \def\bbl@hyphen@i#1#2{%
2245   \bbl@ifunset{bbl@hy@#1#2\@empty}%
2246     {\csname bbl@#1usehyphen\endcsname{\discretionary{#2}{}{#2}}}%
2247     {\csname bbl@hy@#1#2\@empty\endcsname}}
```

The following two commands are used to wrap the "hyphen" and set the behavior of the rest of the word – the version with a single @ is used when further hyphenation is allowed, while that with @@ if no more hyphens are allowed. In both cases, if the hyphen is preceded by a positive space, breaking after the hyphen is disallowed.

---

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

There should not be a discretionary after a hyphen at the beginning of a word, so it is prevented if preceded by a skip. Unfortunately, this does handle cases like "(-suffix)". \nobreak is always preceded by \leavevmode, in case the shorthand starts a paragraph.

```
2248 \def\bbl@usehyphen#1{%
2249   \leavevmode
2250   \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
2251   \nobreak\hskip\z@skip}
2252 \def\bbl@@usehyphen#1{%
2253   \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}
```

The following macro inserts the hyphen char.

```
2254 \def\bbl@hyphenchar{%
2255   \ifnum\hyphenchar\font=\m@ne
2256     \babelnullhyphen
2257   \else
2258     \char\hyphenchar\font
2259   \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.

```
2260 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}{}}}
2261 \def\bbl@hy@@soft{\bbl@@usehyphen{\discretionary{\bbl@hyphenchar}{}{}}}
2262 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
2263 \def\bbl@hy@@hard{\bbl@@usehyphen\bbl@hyphenchar}
2264 \def\bbl@hy@nobreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}}
2265 \def\bbl@hy@@nobreak{\mbox{\bbl@hyphenchar}}
2266 \def\bbl@hy@repeat{%
2267   \bbl@usehyphen{%
2268     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
2269 \def\bbl@hy@@repeat{%
2270   \bbl@@usehyphen{%
2271     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
2272 \def\bbl@hy@empty{\hskip\z@skip}
2273 \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.

```
2274 \def\bbl@disc#1#2{\nobreak\discretionary{#2-}{}{#1}\bbl@allowhyphens}
```

9.10   Multiencoding strings

The aim following commands is to provide a commom interface for strings in several encodings. They also contains several hooks which can be used by luatex and xetex. The code is organized here with pseudo-guards, so we start with the basic commands.

Tools   But first, a couple of tools. The first one makes global a local variable. This is not the best solution, but it works.

```
2275 \bbl@trace{Multiencoding strings}
2276 \def\bbl@toglobal#1{\global\let#1#1}
2277 \def\bbl@recatcode#1{% TODO. Used only once?
2278   \@tempcnta="7F
2279   \def\bbl@tempa{%
2280     \ifnum\@tempcnta>"FF\else
2281       \catcode\@tempcnta=#1\relax
2282       \advance\@tempcnta\@ne
2283       \expandafter\bbl@tempa
```

```
2284        \fi}%
2285    \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).

```
2286 \@ifpackagewith{babel}{nocase}%
2287    {\let\bbl@patchuclc\relax}%
2288    {\def\bbl@patchuclc{%
2289      \global\let\bbl@patchuclc\relax
2290      \g@addto@macro\@uclclist{\reserved@b{\reserved@b\bbl@uclc}}%
2291      \gdef\bbl@uclc##1{%
2292        \let\bbl@encoded\bbl@encoded@uclc
2293        \bbl@ifunset{\languagename @bbl@uclc}% and resumes it
2294          {##1}%
2295          {\let\bbl@tempa##1\relax % Used by LANG@bbl@uclc
2296           \csname\languagename @bbl@uclc\endcsname}%
2297        {\bbl@tolower\@empty}{\bbl@toupper\@empty}}%
2298      \gdef\bbl@tolower{\csname\languagename @bbl@lc\endcsname}%
2299      \gdef\bbl@toupper{\csname\languagename @bbl@uc\endcsname}}}
```

```
2300 ⟨⟨∗More package options⟩⟩ ≡
2301 \DeclareOption{nocase}{}
2302 ⟨⟨/More package options⟩⟩
```

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

```
2303 ⟨⟨∗More package options⟩⟩ ≡
2304 \let\bbl@opt@strings\@nnil % accept strings=value
2305 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
2306 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
2307 \def\BabelStringsDefault{generic}
2308 ⟨⟨/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.

```
2309 \@onlypreamble\StartBabelCommands
2310 \def\StartBabelCommands{%
2311   \begingroup
2312   \bbl@recatcode{11}%
2313   ⟨⟨Macros local to BabelCommands⟩⟩
2314   \def\bbl@provstring##1##2{%
2315     \providecommand##1{##2}%
2316     \bbl@toglobal##1}%
2317   \global\let\bbl@scafter\@empty
2318   \let\StartBabelCommands\bbl@startcmds
2319   \ifx\BabelLanguages\relax
2320      \let\BabelLanguages\CurrentOption
2321   \fi
```

```
2322   \begingroup
2323   \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
2324   \StartBabelCommands}
2325 \def\bbl@startcmds{%
2326   \ifx\bbl@screset\@nnil\else
2327     \bbl@usehooks{stopcommands}{}%
2328   \fi
2329   \endgroup
2330   \begingroup
2331   \@ifstar
2332     {\ifx\bbl@opt@strings\@nnil
2333        \let\bbl@opt@strings\BabelStringsDefault
2334      \fi
2335      \bbl@startcmds@i}%
2336     \bbl@startcmds@i}
2337 \def\bbl@startcmds@i#1#2{%
2338   \edef\bbl@L{\zap@space#1 \@empty}%
2339   \edef\bbl@G{\zap@space#2 \@empty}%
2340   \bbl@startcmds@ii}
2341 \let\bbl@startcommands\StartBabelCommands
```

Parse the encoding info to get the label, input, and font parts.

Select the behavior of \SetString. Thre are two main cases, depending of if there is an optional argument: without it and strings=encoded, strings are defined always; otherwise, they are set only if they are still undefined (ie, fallback values). With labelled blocks and strings=encoded, define the strings, but with another value, define strings only if the current label or font encoding is the value of strings; otherwise (ie, no strings or a block whose label is not in strings=) do nothing.

We presume the current block is not loaded, and therefore set (above) a couple of default values to gobble the arguments. Then, these macros are redefined if necessary according to several parameters.

```
2342 \newcommand\bbl@startcmds@ii[1][\@empty]{%
2343   \let\SetString\@gobbletwo
2344   \let\bbl@stringdef\@gobbletwo
2345   \let\AfterBabelCommands\@gobble
2346   \ifx\@empty#1%
2347     \def\bbl@sc@label{generic}%
2348     \def\bbl@encstring##1##2{%
2349       \ProvideTextCommandDefault##1{##2}%
2350       \bbl@toglobal##1%
2351       \expandafter\bbl@toglobal\csname\string?\string##1\endcsname}%
2352     \let\bbl@sctest\in@true
2353   \else
2354     \let\bbl@sc@charset\space % <- zapped below
2355     \let\bbl@sc@fontenc\space % <-    "        "
2356     \def\bbl@tempa##1=##2\@nil{%
2357       \bbl@csarg\edef{sc@\zap@space##1 \@empty}{##2 }}%
2358     \bbl@vforeach{label=#1}{\bbl@tempa##1\@nil}%
2359     \def\bbl@tempa##1 ##2{% space -> comma
2360       ##1%
2361       \ifx\@empty##2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa##2\fi}%
2362     \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%
2363     \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%
2364     \edef\bbl@sc@charset{\expandafter\zap@space\bbl@sc@charset\@empty}%
2365     \def\bbl@encstring##1##2{%
2366       \bbl@foreach\bbl@sc@fontenc{%
2367         \bbl@ifunset{T@####1}%
2368           {}%
```

```
2369              {\ProvideTextCommand##1{####1}{##2}%
2370               \bbl@toglobal##1%
2371               \expandafter
2372               \bbl@toglobal\csname####1\string##1\endcsname}}}%
2373        \def\bbl@sctest{%
2374           \bbl@xin@{,\bbl@opt@strings,}{,\bbl@sc@label,\bbl@sc@fontenc,}}%
2375        \fi
2376        \ifx\bbl@opt@strings\@nnil        % ie, no strings key -> defaults
2377        \else\ifx\bbl@opt@strings\relax    % ie, strings=encoded
2378           \let\AfterBabelCommands\bbl@aftercmds
2379           \let\SetString\bbl@setstring
2380           \let\bbl@stringdef\bbl@encstring
2381        \else        % ie, strings=value
2382        \bbl@sctest
2383        \ifin@
2384           \let\AfterBabelCommands\bbl@aftercmds
2385           \let\SetString\bbl@setstring
2386           \let\bbl@stringdef\bbl@provstring
2387        \fi\fi\fi
2388        \bbl@scswitch
2389        \ifx\bbl@G\@empty
2390           \def\SetString##1##2{%
2391             \bbl@error{Missing group for string \string##1}%
2392                {You must assign strings to some category, typically\\%
2393                 captions or extras, but you set none}}%
2394        \fi
2395        \ifx\@empty#1%
2396           \bbl@usehooks{defaultcommands}{}%
2397        \else
2398           \@expandtwoargs
2399           \bbl@usehooks{encodedcommands}{{\bbl@sc@charset}{\bbl@sc@fontenc}}%
2400        \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) .

```
2401 \def\bbl@forlang#1#2{%
2402   \bbl@for#1\bbl@L{%
2403     \bbl@xin@{,#1,}{,\BabelLanguages,}%
2404     \ifin@#2\relax\fi}}
2405 \def\bbl@scswitch{%
2406   \bbl@forlang\bbl@tempa{%
2407     \ifx\bbl@G\@empty\else
2408       \ifx\SetString\@gobbletwo\else
2409         \edef\bbl@GL{\bbl@G\bbl@tempa}%
2410         \bbl@xin@{,\bbl@GL,}{,\bbl@screset,}%
2411         \ifin@\else
2412           \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
2413           \xdef\bbl@screset{\bbl@screset,\bbl@GL}%
2414         \fi
2415       \fi
2416     \fi}}
```

119

```
2417 \AtEndOfPackage{%
2418   \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}{}{#2}}}%
2419   \let\bbl@scswitch\relax}
2420 \@onlypreamble\EndBabelCommands
2421 \def\EndBabelCommands{%
2422   \bbl@usehooks{stopcommands}{}%
2423   \endgroup
2424   \endgroup
2425   \bbl@scafter}
2426 \let\bbl@endcommands\EndBabelCommands
```

Now we define commands to be used inside \StartBabelCommands.

Strings The following macro is the actual definition of \SetString when it is "active"
First save the "switcher". Create it if undefined. Strings are defined only if undefined (ie,
like \providescommmand). With the event stringprocess you can preprocess the string by
manipulating the value of \BabelString. If there are several hooks assigned to this event,
preprocessing is done in the same order as defined. Finally, the string is set.

```
2427 \def\bbl@setstring#1#2{%
2428   \bbl@forlang\bbl@tempa{%
2429     \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
2430     \bbl@ifunset{\bbl@LC}% eg, \germanchaptername
2431       {\global\expandafter  % TODO - con \bbl@exp ?
2432        \bbl@add\csname\bbl@G\bbl@tempa\expandafter\endcsname\expandafter
2433          {\expandafter\bbl@scset\expandafter#1\csname\bbl@LC\endcsname}}%
2434       {}%
2435     \def\BabelString{#2}%
2436     \bbl@usehooks{stringprocess}{}%
2437     \expandafter\bbl@stringdef
2438       \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.

```
2439 \ifx\bbl@opt@strings\relax
2440   \def\bbl@scset#1#2{\def#1{\bbl@encoded#2}}
2441   \bbl@patchuclc
2442   \let\bbl@encoded\relax
2443   \def\bbl@encoded@uclc#1{%
2444     \@inmathwarn#1%
2445     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
2446       \expandafter\ifx\csname ?\string#1\endcsname\relax
2447         \TextSymbolUnavailable#1%
2448       \else
2449         \csname ?\string#1\endcsname
2450       \fi
2451     \else
2452       \csname\cf@encoding\string#1\endcsname
2453     \fi}
2454 \else
2455   \def\bbl@scset#1#2{\def#1{#2}}
2456 \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.

120

```
2457 ⟨∗Macros local to BabelCommands⟩ ≡
2458 \def\SetStringLoop##1##2{%
2459     \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
2460     \count@\z@
2461     \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
2462         \advance\count@\@ne
2463         \toks@\expandafter{\bbl@tempa}%
2464         \bbl@exp{%
2465             \\\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
2466             \count@=\the\count@\relax}}}%
2467 ⟨⟨/Macros local to BabelCommands⟩⟩
```

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

```
2468 \def\bbl@aftercmds#1{%
2469   \toks@\expandafter{\bbl@scafter#1}%
2470   \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.

```
2471 ⟨∗Macros local to BabelCommands⟩ ≡
2472   \newcommand\SetCase[3][]{%
2473     \bbl@patchuclc
2474     \bbl@forlang\bbl@tempa{%
2475       \expandafter\bbl@encstring
2476         \csname\bbl@tempa @bbl@uclc\endcsname{\bbl@tempa##1}%
2477       \expandafter\bbl@encstring
2478         \csname\bbl@tempa @bbl@uc\endcsname{##2}%
2479       \expandafter\bbl@encstring
2480         \csname\bbl@tempa @bbl@lc\endcsname{##3}}}%
2481 ⟨⟨/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.

```
2482 ⟨∗Macros local to BabelCommands⟩ ≡
2483   \newcommand\SetHyphenMap[1]{%
2484     \bbl@forlang\bbl@tempa{%
2485       \expandafter\bbl@stringdef
2486         \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
2487 ⟨⟨/Macros local to BabelCommands⟩⟩
```

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

```
2488 \newcommand\BabelLower[2]{% one to one.
2489   \ifnum\lccode#1=#2\else
2490     \babel@savevariable{\lccode#1}%
2491     \lccode#1=#2\relax
2492   \fi}
2493 \newcommand\BabelLowerMM[4]{% many-to-many
2494   \@tempcnta=#1\relax
2495   \@tempcntb=#4\relax
2496   \def\bbl@tempa{%
2497     \ifnum\@tempcnta>#2\else
2498       \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
2499       \advance\@tempcnta#3\relax
2500       \advance\@tempcntb#3\relax
2501       \expandafter\bbl@tempa
```

```
2502        \fi}%
2503    \bbl@tempa}
2504 \newcommand\BabelLowerMO[4]{% many-to-one
2505    \@tempcnta=#1\relax
2506    \def\bbl@tempa{%
2507      \ifnum\@tempcnta>#2\else
2508        \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
2509        \advance\@tempcnta#3
2510        \expandafter\bbl@tempa
2511      \fi}%
2512    \bbl@tempa}
```

The following package options control the behavior of hyphenation mapping.

```
2513 ⟨⟨*More package options⟩⟩ ≡
2514 \DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\z@}
2515 \DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap\@ne}
2516 \DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\tw@}
2517 \DeclareOption{hyphenmap=other}{\chardef\bbl@opt@hyphenmap\thr@@}
2518 \DeclareOption{hyphenmap=other*}{\chardef\bbl@opt@hyphenmap4\relax}
2519 ⟨⟨/More package options⟩⟩
```

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

```
2520 \AtEndOfPackage{%
2521    \ifx\bbl@opt@hyphenmap\@undefined
2522      \bbl@xin@{,}{\bbl@language@opts}%
2523      \chardef\bbl@opt@hyphenmap\ifin@4\else\@ne\fi
2524    \fi}
```

9.11 Macros common to a number of languages

\set@low@box The following macro is used to lower quotes to the same level as the comma. It prepares its argument in box register 0.

```
2525 \bbl@trace{Macros related to glyphs}
2526 \def\set@low@box#1{\setbox\tw@\hbox{,}\setbox\z@\hbox{#1}%
2527    \dimen\z@\ht\z@ \advance\dimen\z@ -\ht\tw@%
2528    \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.

```
2529 \def\save@sf@q#1{\leavevmode
2530    \begingroup
2531      \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
2532    \endgroup}
```

9.12 Making glyphs available

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

9.12.1 Quotation marks

\quotedblbase In the T1 encoding the opening double quote at the baseline is available as a separate character, accessible via \quotedblbase. In the OT1 encoding it is not available, therefore we make it available by lowering the normal open quote character to the baseline.

```
2533 \ProvideTextCommand{\quotedblbase}{OT1}{%
2534    \save@sf@q{\set@low@box{\textquotedblright\/}%
2535      \box\z@\kern-.04em\bbl@allowhyphens}}
```

122

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

```
2536 \ProvideTextCommandDefault{\quotedblbase}{%
2537   \UseTextSymbol{OT1}{\quotedblbase}}
```

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

```
2538 \ProvideTextCommand{\quotesinglbase}{OT1}{%
2539   \save@sf@q{\set@low@box{\textquoteright\/}%
2540     \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.

```
2541 \ProvideTextCommandDefault{\quotesinglbase}{%
2542   \UseTextSymbol{OT1}{\quotesinglbase}}
```

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

```
2543 \ProvideTextCommand{\guillemetleft}{OT1}{%
2544   \ifmmode
2545     \ll
2546   \else
2547     \save@sf@q{\nobreak
2548       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2549   \fi}
2550 \ProvideTextCommand{\guillemetright}{OT1}{%
2551   \ifmmode
2552     \gg
2553   \else
2554     \save@sf@q{\nobreak
2555       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2556   \fi}
2557 \ProvideTextCommand{\guillemotleft}{OT1}{%
2558   \ifmmode
2559     \ll
2560   \else
2561     \save@sf@q{\nobreak
2562       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2563   \fi}
2564 \ProvideTextCommand{\guillemotright}{OT1}{%
2565   \ifmmode
2566     \gg
2567   \else
2568     \save@sf@q{\nobreak
2569       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2570   \fi}
```

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

```
2571 \ProvideTextCommandDefault{\guillemetleft}{%
2572   \UseTextSymbol{OT1}{\guillemetleft}}
2573 \ProvideTextCommandDefault{\guillemetright}{%
2574   \UseTextSymbol{OT1}{\guillemetright}}
2575 \ProvideTextCommandDefault{\guillemotleft}{%
2576   \UseTextSymbol{OT1}{\guillemotleft}}
2577 \ProvideTextCommandDefault{\guillemotright}{%
2578   \UseTextSymbol{OT1}{\guillemotright}}
```

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

```
2579 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2580   \ifmmode
2581     <%
2582   \else
2583     \save@sf@q{\nobreak
2584       \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
2585   \fi}
2586 \ProvideTextCommand{\guilsinglright}{OT1}{%
2587   \ifmmode
2588     >%
2589   \else
2590     \save@sf@q{\nobreak
2591       \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
2592   \fi}
```

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

```
2593 \ProvideTextCommandDefault{\guilsinglleft}{%
2594   \UseTextSymbol{OT1}{\guilsinglleft}}
2595 \ProvideTextCommandDefault{\guilsinglright}{%
2596   \UseTextSymbol{OT1}{\guilsinglright}}
```

9.12.2 Letters

\ij The dutch language uses the letter 'ij'. It is available in T1 encoded fonts, but not in the OT1
\IJ encoded fonts. Therefore we fake it for the OT1 encoding.

```
2597 \DeclareTextCommand{\ij}{OT1}{%
2598   i\kern-0.02em\bbl@allowhyphens j}
2599 \DeclareTextCommand{\IJ}{OT1}{%
2600   I\kern-0.02em\bbl@allowhyphens J}
2601 \DeclareTextCommand{\ij}{T1}{\char188}
2602 \DeclareTextCommand{\IJ}{T1}{\char156}
```

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

```
2603 \ProvideTextCommandDefault{\ij}{%
2604   \UseTextSymbol{OT1}{\ij}}
2605 \ProvideTextCommandDefault{\IJ}{%
2606   \UseTextSymbol{OT1}{\IJ}}
```

\dj The croatian language needs the letters \dj and \DJ; they are available in the T1 encoding,
\DJ but not in the OT1 encoding by default.
     Some code to construct these glyphs for the OT1 encoding was made available to me by
     Stipčević Mario, (stipcevic@olimp.irb.hr).

```
2607 \def\crrtic@{\hrule height0.1ex width0.3em}
2608 \def\crttic@{\hrule height0.1ex width0.33em}
2609 \def\ddj@{%
2610   \setbox0\hbox{d}\dimen@=\ht0
2611   \advance\dimen@1ex
2612   \dimen@.45\dimen@
2613   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2614   \advance\dimen@ii.5ex
2615   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2616 \def\DDJ@{%
2617   \setbox0\hbox{D}\dimen@=.55\ht0
2618   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
```

```
2619    \advance\dimen@ii.15ex %                correction for the dash position
2620    \advance\dimen@ii-.15\fontdimen7\font %     correction for cmtt font
2621    \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
2622    \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
2623 %
2624 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2625 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}
```

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

```
2626 \ProvideTextCommandDefault{\dj}{%
2627    \UseTextSymbol{OT1}{\dj}}
2628 \ProvideTextCommandDefault{\DJ}{%
2629    \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.

```
2630 \DeclareTextCommand{\SS}{OT1}{SS}
2631 \ProvideTextCommandDefault{\SS}{\UseTextSymbol{OT1}{\SS}}
```

9.12.3   Shorthands for quotation marks

Shorthands are provided for a number of different quotation marks, which make them usable both outside and inside mathmode. They are defined with \ProvideTextCommandDefault, but this is very likely not required because their definitions are based on encoding-dependent macros.

\glq   The 'german' single quotes.
\grq
```
2632 \ProvideTextCommandDefault{\glq}{%
2633    \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}
```

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

```
2634 \ProvideTextCommand{\grq}{T1}{%
2635    \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}
2636 \ProvideTextCommand{\grq}{TU}{%
2637    \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
2638 \ProvideTextCommand{\grq}{OT1}{%
2639    \save@sf@q{\kern-.0125em
2640      \textormath{\textquoteleft}{\mbox{\textquoteleft}}%
2641      \kern.07em\relax}}
2642 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}\grq}
```

\glqq   The 'german' double quotes.
\grqq
```
2643 \ProvideTextCommandDefault{\glqq}{%
2644    \textormath{\quotedblbase}{\mbox{\quotedblbase}}}
```

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

```
2645 \ProvideTextCommand{\grqq}{T1}{%
2646    \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2647 \ProvideTextCommand{\grqq}{TU}{%
2648    \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2649 \ProvideTextCommand{\grqq}{OT1}{%
2650    \save@sf@q{\kern-.07em
2651      \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}%
2652      \kern.07em\relax}}
2653 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}
```

125

\flq   The 'french' single guillemets.
\frq

2654 \ProvideTextCommandDefault{\flq}{%
2655   \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
2656 \ProvideTextCommandDefault{\frq}{%
2657   \textormath{\guilsinglright}{\mbox{\guilsinglright}}}

\flqq   The 'french' double guillemets.
\frqq

2658 \ProvideTextCommandDefault{\flqq}{%
2659   \textormath{\guillemetleft}{\mbox{\guillemetleft}}}
2660 \ProvideTextCommandDefault{\frqq}{%
2661   \textormath{\guillemetright}{\mbox{\guillemetright}}}

9.12.4   Umlauts and tremas

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

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

2662 \def\umlauthigh{%
2663   \def\bbl@umlauta##1{\leavevmode\bgroup%
2664   \expandafter\accent\csname\f@encoding dqpos\endcsname
2665   ##1\bbl@allowhyphens\egroup}%
2666   \let\bbl@umlaute\bbl@umlauta}
2667 \def\umlautlow{%
2668   \def\bbl@umlauta{\protect\lower@umlaut}}
2669 \def\umlautelow{%
2670   \def\bbl@umlaute{\protect\lower@umlaut}}
2671 \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.

2672 \expandafter\ifx\csname U@D\endcsname\relax
2673   \csname newdimen\endcsname\U@D
2674 \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.

2675 \def\lower@umlaut#1{%
2676   \leavevmode\bgroup
2677   \U@D 1ex%
2678   {\setbox\z@\hbox{%
2679   \expandafter\char\csname\f@encoding dqpos\endcsname}%
2680   \dimen@ -.45ex\advance\dimen@\ht\z@
2681   \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2682   \expandafter\accent\csname\f@encoding dqpos\endcsname

126

```
2683     \fontdimen5\font\U@D #1%
2684   \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).

```
2685 \AtBeginDocument{%
2686   \DeclareTextCompositeCommand{\"}{OT1}{a}{\bbl@umlauta{a}}%
2687   \DeclareTextCompositeCommand{\"}{OT1}{e}{\bbl@umlaute{e}}%
2688   \DeclareTextCompositeCommand{\"}{OT1}{i}{\bbl@umlaute{\i}}%
2689   \DeclareTextCompositeCommand{\"}{OT1}{\i}{\bbl@umlaute{\i}}%
2690   \DeclareTextCompositeCommand{\"}{OT1}{o}{\bbl@umlauta{o}}%
2691   \DeclareTextCompositeCommand{\"}{OT1}{u}{\bbl@umlauta{u}}%
2692   \DeclareTextCompositeCommand{\"}{OT1}{A}{\bbl@umlauta{A}}%
2693   \DeclareTextCompositeCommand{\"}{OT1}{E}{\bbl@umlaute{E}}%
2694   \DeclareTextCompositeCommand{\"}{OT1}{I}{\bbl@umlaute{I}}%
2695   \DeclareTextCompositeCommand{\"}{OT1}{O}{\bbl@umlauta{O}}%
2696   \DeclareTextCompositeCommand{\"}{OT1}{U}{\bbl@umlauta{U}}%
2697 }
```

Finally, make sure the default hyphenrules are defined (even if empty).

```
2698 \ifx\l@english\@undefined
2699   \chardef\l@english\z@
2700 \fi
```

9.13   Layout

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

```
2701 \bbl@trace{Bidi layout}
2702 \providecommand\IfBabelLayout[3]{#3}%
2703 \newcommand\BabelPatchSection[1]{%
2704   \@ifundefined{#1}{}{%
2705     \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
2706     \@namedef{#1}{%
2707       \@ifstar{\bbl@presec@s{#1}}%
2708               {\@dblarg{\bbl@presec@x{#1}}}}}}
2709 \def\bbl@presec@x#1[#2]#3{%
2710   \bbl@exp{%
2711     \\\select@language@x{\bbl@main@language}%
2712     \\\bbl@cs{sspre@#1}%
2713     \\\bbl@cs{ss@#1}%
2714       [\\\foreignlanguage{\languagename}{\unexpanded{#2}}]%
2715       {\\\foreignlanguage{\languagename}{\unexpanded{#3}}}%
2716     \\\select@language@x{\languagename}}}
2717 \def\bbl@presec@s#1#2{%
2718   \bbl@exp{%
2719     \\\select@language@x{\bbl@main@language}%
2720     \\\bbl@cs{sspre@#1}%
2721     \\\bbl@cs{ss@#1}*%
2722       {\\\foreignlanguage{\languagename}{\unexpanded{#2}}}%
2723     \\\select@language@x{\languagename}}}
```

```
2724 \IfBabelLayout{sectioning}%
2725   {\BabelPatchSection{part}%
2726    \BabelPatchSection{chapter}%
2727    \BabelPatchSection{section}%
2728    \BabelPatchSection{subsection}%
2729    \BabelPatchSection{subsubsection}%
2730    \BabelPatchSection{paragraph}%
2731    \BabelPatchSection{subparagraph}%
2732    \def\babel@toc#1{%
2733      \select@language@x{\bbl@main@language}}}{}
2734 \IfBabelLayout{captions}%
2735   {\BabelPatchSection{caption}}{}
```

9.14   Load engine specific macros

```
2736 \bbl@trace{Input engine specific macros}
2737 \ifcase\bbl@engine
2738   \input txtbabel.def
2739 \or
2740   \input luababel.def
2741 \or
2742   \input xebabel.def
2743 \fi
```

9.15   Creating and modifying languages

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

```
2744 \bbl@trace{Creating languages and reading ini files}
2745 \newcommand\babelprovide[2][]{%
2746   \let\bbl@savelangname\languagename
2747   \edef\bbl@savelocaleid{\the\localeid}%
2748   % Set name and locale id
2749   \edef\languagename{#2}%
2750   % \global\@namedef{bbl@lcname@#2}{#2}%
2751   \bbl@id@assign
2752   \let\bbl@KVP@captions\@nil
2753   \let\bbl@KVP@import\@nil
2754   \let\bbl@KVP@main\@nil
2755   \let\bbl@KVP@script\@nil
2756   \let\bbl@KVP@language\@nil
2757   \let\bbl@KVP@hyphenrules\@nil  % only for provide@new
2758   \let\bbl@KVP@mapfont\@nil
2759   \let\bbl@KVP@maparabic\@nil
2760   \let\bbl@KVP@mapdigits\@nil
2761   \let\bbl@KVP@intraspace\@nil
2762   \let\bbl@KVP@intrapenalty\@nil
2763   \let\bbl@KVP@onchar\@nil
2764   \let\bbl@KVP@alph\@nil
2765   \let\bbl@KVP@Alph\@nil
2766   \let\bbl@KVP@info\@nil % Ignored with import? Or error/warning?
2767   \bbl@forkv{#1}{%  TODO - error handling
2768     \in@{/}{##1}%
2769     \ifin@
2770       \bbl@renewinikey##1\@@{##2}%
2771     \else
2772       \bbl@csarg\def{KVP@##1}{##2}%
```

```
2773      \fi}%
2774  % == import, captions ==
2775  \ifx\bbl@KVP@import\@nil\else
2776    \bbl@exp{\\\bbl@ifblank{\bbl@KVP@import}}%
2777      {\ifx\bbl@initoload\relax
2778        \begingroup
2779          \def\BabelBeforeIni##1##2{\gdef\bbl@KVP@import{##1}\endinput}%
2780          \InputIfFileExists{babel-#2.tex}{}{}%
2781        \endgroup
2782      \else
2783        \xdef\bbl@KVP@import{\bbl@initoload}%
2784      \fi}%
2785      {}%
2786  \fi
2787  \ifx\bbl@KVP@captions\@nil
2788    \let\bbl@KVP@captions\bbl@KVP@import
2789  \fi
2790  % Load ini
2791  \bbl@ifunset{date#2}%
2792    {\bbl@provide@new{#2}}%
2793    {\bbl@ifblank{#1}%
2794      {\bbl@error
2795        {If you want to modify `#2' you must tell how in\\%
2796         the optional argument. See the manual for the\\%
2797         available options.}%
2798        {Use this macro as documented}}%
2799      {\bbl@provide@renew{#2}}}%
2800  % Post tasks
2801  \bbl@exp{\\\babelensure[exclude=\\\today]{#2}}%
2802  \bbl@ifunset{bbl@ensure@\languagename}%
2803    {\bbl@exp{%
2804      \\\DeclareRobustCommand\<bbl@ensure@\languagename>[1]{%
2805        \\\foreignlanguage{\languagename}%
2806        {####1}}}}%
2807    {}%
2808  \bbl@exp{%
2809    \\\bbl@toglobal\<bbl@ensure@\languagename>%
2810    \\\bbl@toglobal\<bbl@ensure@\languagename\space>}%
2811  % At this point all parameters are defined if 'import'. Now we
2812  % execute some code depending on them. But what about if nothing was
2813  % imported? We just load the very basic parameters: ids and a few
2814  % more.
2815  \bbl@ifunset{bbl@lname@#2}%  TODO. Duplicated
2816    {\def\BabelBeforeIni##1##2{%
2817      \begingroup
2818        \catcode`\[=12 \catcode`\]=12 \catcode`\==12  \catcode`\;=12 %
2819        \let\bbl@ini@captions@aux\@gobbletwo
2820        \def\bbl@inidate ####1.####2.####3.####4\relax ####5####6{}%
2821        \bbl@read@ini{##1}{basic data}%
2822        \bbl@exportkey{chrng}{characters.ranges}{}%
2823        \bbl@exportkey{dgnat}{numbers.digits.native}{}%
2824        \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
2825        \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
2826        \bbl@exportkey{lfthm}{typography.lefthyphenmin}{2}%
2827        \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
2828        \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
2829        \bbl@exportkey{hyoth}{typography.hyphenate.other}{}%
2830        \bbl@exportkey{intsp}{typography.intraspace}{}%
2831        \ifx\bbl@initoload\relax\endinput\fi
```

```
2832        \endgroup}%
2833      \begingroup       % boxed, to avoid extra spaces:
2834        \ifx\bbl@initoload\relax
2835          \setbox\z@\hbox{\InputIfFileExists{babel-#2.tex}{}{}}%
2836        \else
2837          \setbox\z@\hbox{\BabelBeforeIni{\bbl@initoload}{}}%
2838        \fi
2839      \endgroup}%
2840    {}%
2841  % == script, language ==
2842  % Override the values from ini or defines them
2843  \ifx\bbl@KVP@script\@nil\else
2844    \bbl@csarg\edef{sname@#2}{\bbl@KVP@script}%
2845  \fi
2846  \ifx\bbl@KVP@language\@nil\else
2847    \bbl@csarg\edef{lname@#2}{\bbl@KVP@language}%
2848  \fi
2849   % == onchar ==
2850  \ifx\bbl@KVP@onchar\@nil\else
2851    \bbl@luahyphenate
2852    \directlua{
2853      if Babel.locale_mapped == nil then
2854        Babel.locale_mapped = true
2855        Babel.linebreaking.add_before(Babel.locale_map)
2856        Babel.loc_to_scr = {}
2857        Babel.chr_to_loc = Babel.chr_to_loc or {}
2858      end}%
2859    \bbl@xin@{ ids }{ \bbl@KVP@onchar\space}%
2860    \ifin@
2861      \ifx\bbl@starthyphens\@undefined % Needed if no explicit selection
2862        \AddBabelHook{babel-onchar}{beforestart}{{\bbl@starthyphens}}%
2863      \fi
2864      \bbl@exp{\\\bbl@add\\\bbl@starthyphens
2865        {\\\bbl@patterns@lua{\languagename}}}%
2866      % TODO - error/warning if no script
2867      \directlua{
2868        if Babel.script_blocks['\bbl@cl{sbcp}'] then
2869          Babel.loc_to_scr[\the\localeid] =
2870            Babel.script_blocks['\bbl@cl{sbcp}']
2871          Babel.locale_props[\the\localeid].lc = \the\localeid\space
2872          Babel.locale_props[\the\localeid].lg = \the\@nameuse{l@\languagename}\space
2873        end
2874      }%
2875    \fi
2876    \bbl@xin@{ fonts }{ \bbl@KVP@onchar\space}%
2877    \ifin@
2878      \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
2879      \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
2880      \directlua{
2881        if Babel.script_blocks['\bbl@cl{sbcp}'] then
2882          Babel.loc_to_scr[\the\localeid] =
2883            Babel.script_blocks['\bbl@cl{sbcp}']
2884        end}%
2885      \ifx\bbl@mapselect\@undefined
2886        \AtBeginDocument{%
2887          \expandafter\bbl@add\csname selectfont \endcsname{{\bbl@mapselect}}%
2888          {\selectfont}}%
2889        \def\bbl@mapselect{%
2890          \let\bbl@mapselect\relax
```

130

```
2891          \edef\bbl@prefontid{\fontid\font}}%
2892        \def\bbl@mapdir##1{%
2893          {\def\languagename{##1}%
2894           \let\bbl@ifrestoring\@firstoftwo % To avoid font warning
2895           \bbl@switchfont
2896           \directlua{
2897             Babel.locale_props[\the\csname bbl@id@@##1\endcsname]%
2898                         ['/\bbl@prefontid'] = \fontid\font\space}}}%
2899      \fi
2900      \bbl@exp{\\\bbl@add\\\bbl@mapselect{\\\bbl@mapdir{\languagename}}}%
2901    \fi
2902    % TODO - catch non-valid values
2903    \fi
2904  % == mapfont ==
2905  % For bidi texts, to switch the font based on direction
2906  \ifx\bbl@KVP@mapfont\@nil\else
2907    \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}{}%
2908      {\bbl@error{Option `\bbl@KVP@mapfont' unknown for\\%
2909                  mapfont. Use `direction'.%
2910                  {See the manual for details.}}}%
2911    \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
2912    \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}%
2913    \ifx\bbl@mapselect\@undefined
2914      \AtBeginDocument{%
2915        \expandafter\bbl@add\csname selectfont \endcsname{{\bbl@mapselect}}%
2916        {\selectfont}}%
2917      \def\bbl@mapselect{%
2918        \let\bbl@mapselect\relax
2919        \edef\bbl@prefontid{\fontid\font}}%
2920      \def\bbl@mapdir##1{%
2921        {\def\languagename{##1}%
2922         \let\bbl@ifrestoring\@firstoftwo % avoid font warning
2923         \bbl@switchfont
2924         \directlua{Babel.fontmap
2925           [\the\csname bbl@wdir@##1\endcsname]%
2926           [\bbl@prefontid]=\fontid\font}}}%
2927    \fi
2928    \bbl@exp{\\\bbl@add\\\bbl@mapselect{\\\bbl@mapdir{\languagename}}}%
2929  \fi
2930  % == intraspace, intrapenalty ==
2931  % For CJK, East Asian, Southeast Asian, if interspace in ini
2932  \ifx\bbl@KVP@intraspace\@nil\else % We can override the ini or set
2933    \bbl@csarg\edef{intsp@#2}{\bbl@KVP@intraspace}%
2934  \fi
2935  \bbl@provide@intraspace
2936  % == hyphenate.other ==
2937  \bbl@ifunset{bbl@hyoth@\languagename}{}%
2938    {\bbl@csarg\bbl@replace{hyoth@\languagename}{ }{,}%
2939     \bbl@startcommands*{\languagename}{}%
2940       \bbl@csarg\bbl@foreach{hyoth@\languagename}{%
2941         \ifcase\bbl@engine
2942           \ifnum##1<257
2943             \SetHyphenMap\BabelLower{##1}{##1}}%
2944           \fi
2945         \else
2946           \SetHyphenMap\BabelLower{##1}{##1}%
2947         \fi}%
2948     \bbl@endcommands}%
2949  % == maparabic ==
```

131

```
2950   % Native digits, if provided in ini (TeX level, xe and lua)
2951   \ifcase\bbl@engine\else
2952     \bbl@ifunset{bbl@dgnat@\languagename}{}%
2953       {\expandafter\ifx\csname bbl@dgnat@\languagename\endcsname\@empty\else
2954         \expandafter\expandafter\expandafter
2955         \bbl@setdigits\csname bbl@dgnat@\languagename\endcsname
2956         \ifx\bbl@KVP@maparabic\@nil\else
2957           \ifx\bbl@latinarabic\@undefined
2958             \expandafter\let\expandafter\@arabic
2959               \csname bbl@counter@\languagename\endcsname
2960           \else    % ie, if layout=counters, which redefines \@arabic
2961             \expandafter\let\expandafter\bbl@latinarabic
2962               \csname bbl@counter@\languagename\endcsname
2963           \fi
2964         \fi
2965       \fi}%
2966   \fi
2967   % == mapdigits ==
2968   % Native digits (lua level).
2969   \ifodd\bbl@engine
2970     \ifx\bbl@KVP@mapdigits\@nil\else
2971       \bbl@ifunset{bbl@dgnat@\languagename}{}%
2972         {\RequirePackage{luatexbase}%
2973          \bbl@activate@preotf
2974          \directlua{
2975            Babel = Babel or {}  %%% -> presets in luababel
2976            Babel.digits_mapped = true
2977            Babel.digits = Babel.digits or {}
2978            Babel.digits[\the\localeid] =
2979              table.pack(string.utfvalue('\bbl@cl{dgnat}'))
2980            if not Babel.numbers then
2981              function Babel.numbers(head)
2982                local LOCALE = luatexbase.registernumber'bbl@attr@locale'
2983                local GLYPH = node.id'glyph'
2984                local inmath = false
2985                for item in node.traverse(head) do
2986                  if not inmath and item.id == GLYPH then
2987                    local temp = node.get_attribute(item, LOCALE)
2988                    if Babel.digits[temp] then
2989                      local chr = item.char
2990                      if chr > 47 and chr < 58 then
2991                        item.char = Babel.digits[temp][chr-47]
2992                      end
2993                    end
2994                  elseif item.id == node.id'math' then
2995                    inmath = (item.subtype == 0)
2996                  end
2997                end
2998                return head
2999              end
3000            end
3001         }}%
3002     \fi
3003   \fi
3004   % == alph, Alph ==
3005   % What if extras<lang> contains a \babel@save\@alph? It won't be
3006   % restored correctly when exiting the language, so we ignore
3007   % this change with the \bbl@alph@saved trick.
3008   \ifx\bbl@KVP@alph\@nil\else
```

132

```
3009     \toks@\expandafter\expandafter\expandafter{%
3010       \csname extras\languagename\endcsname}%
3011     \bbl@exp{%
3012       \def\<extras\languagename>{%
3013         \let\\\bbl@alph@saved\\\@alph
3014         \the\toks@
3015         \let\\\@alph\\\bbl@alph@saved
3016         \\\babel@save\\\@alph
3017         \let\\\@alph\<bbl@cntr@\bbl@KVP@alph @\languagename>}}%
3018   \fi
3019   \ifx\bbl@KVP@Alph\@nil\else
3020     \toks@\expandafter\expandafter\expandafter{%
3021       \csname extras\languagename\endcsname}%
3022     \bbl@exp{%
3023       \def\<extras\languagename>{%
3024         \let\\\bbl@Alph@saved\\\@Alph
3025         \the\toks@
3026         \let\\\@Alph\\\bbl@Alph@saved
3027         \\\babel@save\\\@Alph
3028         \let\\\@Alph\<bbl@cntr@\bbl@KVP@Alph @\languagename>}}%
3029   \fi
3030   % == require.babel in ini ==
3031   % To load or reaload the babel-*.tex, if require.babel in ini
3032   \bbl@ifunset{bbl@rqtex@\languagename}{}%
3033     {\expandafter\ifx\csname bbl@rqtex@\languagename\endcsname\@empty\else
3034        \let\BabelBeforeIni\@gobbletwo
3035        \chardef\atcatcode=\catcode`\@
3036        \catcode`\@=11\relax
3037        \InputIfFileExists{babel-\bbl@cs{rqtex@\languagename}.tex}{}{}%
3038        \catcode`\@=\atcatcode
3039        \let\atcatcode\relax
3040      \fi}%
3041   % == main ==
3042   \ifx\bbl@KVP@main\@nil  % Restore only if not 'main'
3043     \let\languagename\bbl@savelangname
3044     \chardef\localeid\bbl@savelocaleid\relax
3045   \fi}
```

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.

```
3046 \def\bbl@setdigits#1#2#3#4#5{%
3047   \bbl@exp{%
3048     \def\<\languagename digits>####1{%        ie, \langdigits
3049       \<bbl@digits@\languagename>####1\\\@nil}%
3050     \def\<\languagename counter>####1{%        ie, \langcounter
3051       \\\expandafter\<bbl@counter@\languagename>%
3052       \\\csname c@####1\endcsname}%
3053     \def\<bbl@counter@\languagename>####1{% ie, \bbl@counter@lang
3054       \\\expandafter\<bbl@digits@\languagename>%
3055       \\\number####1\\\@nil}}%
3056   \def\bbl@tempa##1##2##3##4##5{%
3057     \bbl@exp{%    Wow, quite a lot of hashes! :-(
3058       \def\<bbl@digits@\languagename>########1{%
3059         \\\ifx########1\\\@nil              % ie, \bbl@digits@lang
3060         \\\else
3061           \\\ifx0########1#1%
3062           \\\else\\\ifx1########1#2%
3063           \\\else\\\ifx2########1#3%
3064           \\\else\\\ifx3########1#4%
```

133

```
3065          \\\else\\\ifx4########1#5%
3066          \\\else\\\ifx5########1##1%
3067          \\\else\\\ifx6########1##2%
3068          \\\else\\\ifx7########1##3%
3069          \\\else\\\ifx8########1##4%
3070          \\\else\\\ifx9########1##5%
3071          \\\else########1%
3072          \\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi\\\fi
3073          \\\expandafter\<bbl@digits@\languagename>%
3074        \\\fi}}}%
3075    \bbl@tempa}
```

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

```
3076 \def\bbl@provide@new#1{%
3077    \@namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
3078    \@namedef{extras#1}{}%
3079    \@namedef{noextras#1}{}%
3080    \bbl@startcommands*{#1}{captions}%
3081      \ifx\bbl@KVP@captions\@nil %        and also if import, implicit
3082        \def\bbl@tempb##1{%                elt for \bbl@captionslist
3083          \ifx##1\@empty\else
3084            \bbl@exp{%
3085              \\\SetString\\##1{%
3086                \\\bbl@nocaption{\bbl@stripslash##1}{#1\bbl@stripslash##1}}}%
3087            \expandafter\bbl@tempb
3088          \fi}%
3089        \expandafter\bbl@tempb\bbl@captionslist\@empty
3090      \else
3091        \ifx\bbl@initoload\relax
3092          \bbl@read@ini{\bbl@KVP@captions}{data}%  Here letters cat = 11
3093        \else
3094          \bbl@read@ini{\bbl@initoload}{data}%  Here all letters cat = 11
3095        \fi
3096        \bbl@after@ini
3097        \bbl@savestrings
3098      \fi
3099    \StartBabelCommands*{#1}{date}%
3100      \ifx\bbl@KVP@import\@nil
3101        \bbl@exp{%
3102          \\\SetString\\\today{\\\bbl@nocaption{today}{#1today}}}%
3103      \else
3104        \bbl@savetoday
3105        \bbl@savedate
3106      \fi
3107    \bbl@endcommands
3108    \bbl@ifunset{bbl@lname@#1}%        TODO. Duplicated
3109      {\def\BabelBeforeIni##1##2{%
3110        \begingroup
3111          \catcode`\[=12 \catcode`\]=12 \catcode`\==12  \catcode`\;=12 %
3112          \let\bbl@ini@captions@aux\@gobbletwo
3113          \def\bbl@inidate ####1.####2.####3.####4\relax ####5####6{}%
3114          \bbl@read@ini{##1}{basic data}%
3115          \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
3116          \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
3117          \bbl@exportkey{lfthm}{typography.lefthyphenmin}{2}%
3118          \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
3119          \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
3120          \bbl@exportkey{hyoth}{typography.hyphenate.other}{}%
3121          \bbl@exportkey{intsp}{typography.intraspace}{}%
```

```
3122        \bbl@exportkey{chrng}{characters.ranges}{}%
3123        \bbl@exportkey{dgnat}{numbers.digits.native}{}%
3124        \ifx\bbl@initoload\relax\endinput\fi
3125      \endgroup}%
3126    \begingroup        % boxed, to avoid extra spaces:
3127      \ifx\bbl@initoload\relax
3128        \setbox\z@\hbox{\InputIfFileExists{babel-#1.tex}{}{}}%
3129      \else
3130        \setbox\z@\hbox{\BabelBeforeIni{\bbl@initoload}{}}%
3131      \fi
3132    \endgroup}%
3133    {}%
3134  \bbl@exp{%
3135    \gdef\<#1hyphenmins>{%
3136      {\bbl@ifunset{bbl@lfthm@#1}{2}{\bbl@cs{lfthm@#1}}}%
3137      {\bbl@ifunset{bbl@rgthm@#1}{3}{\bbl@cs{rgthm@#1}}}}}%
3138  \bbl@provide@hyphens{#1}%
3139  \ifx\bbl@KVP@main\@nil\else
3140    \expandafter\main@language\expandafter{#1}%
3141  \fi}
3142 \def\bbl@provide@renew#1{%
3143  \ifx\bbl@KVP@captions\@nil\else
3144    \StartBabelCommands*{#1}{captions}%
3145      \bbl@read@ini{\bbl@KVP@captions}{data}%   Here all letters cat = 11
3146      \bbl@after@ini
3147      \bbl@savestrings
3148    \EndBabelCommands
3149  \fi
3150  \ifx\bbl@KVP@import\@nil\else
3151    \StartBabelCommands*{#1}{date}%
3152      \bbl@savetoday
3153      \bbl@savedate
3154    \EndBabelCommands
3155  \fi
3156  % == hyphenrules ==
3157    \bbl@provide@hyphens{#1}}
```

The hyphenrules option is handled with an auxiliary macro.

```
3158 \def\bbl@provide@hyphens#1{%
3159  \let\bbl@tempa\relax
3160  \ifx\bbl@KVP@hyphenrules\@nil\else
3161    \bbl@replace\bbl@KVP@hyphenrules{ }{,}%
3162    \bbl@foreach\bbl@KVP@hyphenrules{%
3163      \ifx\bbl@tempa\relax    % if not yet found
3164        \bbl@ifsamestring{##1}{+}%
3165          {{\bbl@exp{\\\addlanguage\<l@##1>}}}%
3166          {}%
3167        \bbl@ifunset{l@##1}%
3168          {}%
3169          {\bbl@exp{\let\bbl@tempa\<l@##1>}}%
3170      \fi}%
3171  \fi
3172  \ifx\bbl@tempa\relax %        if no opt or no language in opt found
3173    \ifx\bbl@KVP@import\@nil
3174      \ifx\bbl@initoload\relax\else
3175        \bbl@exp{%                    and hyphenrules is not empty
3176          \\\bbl@ifblank{\bbl@cs{hyphr@#1}}%
3177            {}%
3178            {\let\\\bbl@tempa\<l@\bbl@cl{hyphr}>}}%
```

```
3179         \fi
3180     \else % if importing
3181       \bbl@exp{%                        and hyphenrules is not empty
3182         \\\bbl@ifblank{\bbl@cs{hyphr@#1}}%
3183           {}%
3184           {\let\\\bbl@tempa\<l@\bbl@cl{hyphr}>}}%
3185       \fi
3186   \fi
3187   \bbl@ifunset{bbl@tempa}%         ie, relax or undefined
3188     {\bbl@ifunset{l@#1}%            no hyphenrules found - fallback
3189       {\bbl@exp{\\\adddialect\<l@#1>\language}}%
3190       {}}%                            so, l@<lang> is ok - nothing to do
3191     {\bbl@exp{\\\adddialect\<l@#1>\bbl@tempa}}}% found in opt list or ini
3192
```

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.

```
3193 \ifx\bbl@readstream\@undefined
3194   \csname newread\endcsname\bbl@readstream
3195 \fi
3196 \def\bbl@inipreread#1=#2\@@{%
3197   \bbl@trim@def\bbl@tempa{#1}% Redundant below !!
3198   \bbl@trim\toks@{#2}%
3199   % Move trims here ??
3200   \bbl@ifunset{bbl@KVP@\bbl@section/\bbl@tempa}%
3201     {\bbl@exp{%
3202       \\\g@addto@macro\\\bbl@inidata{%
3203         \\\bbl@elt{\bbl@section}{\bbl@tempa}{\the\toks@}}}%
3204     \expandafter\bbl@inireader\bbl@tempa=#2\@@}%
3205     {}}%
3206 \def\bbl@read@ini#1#2{%
3207   \bbl@csarg\edef{lini@\languagename}{#1}%
3208   \openin\bbl@readstream=babel-#1.ini
3209   \ifeof\bbl@readstream
3210     \bbl@error
3211       {There is no ini file for the requested language\\%
3212        (#1). Perhaps you misspelled it or your installation\\%
3213        is not complete.}%
3214       {Fix the name or reinstall babel.}%
3215   \else
3216     \bbl@exp{\def\\\bbl@inidata{\\\bbl@elt{identificacion}{tag.ini}{#1}}}%
3217     \let\bbl@section\@empty
3218     \let\bbl@savestrings\@empty
3219     \let\bbl@savetoday\@empty
3220     \let\bbl@savedate\@empty
3221     \let\bbl@inireader\bbl@iniskip
3222     \bbl@info{Importing #2 for \languagename\\%
3223             from babel-#1.ini. Reported}%
3224     \loop
3225     \if T\ifeof\bbl@readstream F\fi T\relax % Trick, because inside \loop
3226       \endlinechar\m@ne
3227       \read\bbl@readstream to \bbl@line
3228       \endlinechar`\^^M
3229       \ifx\bbl@line\@empty\else
3230         \expandafter\bbl@iniline\bbl@line\bbl@iniline
3231       \fi
3232     \repeat
3233     \bbl@foreach\bbl@renewlist{%
3234       \bbl@ifunset{bbl@renew@##1}{}{\bbl@inisec[##1]\@@}}%
```

```
3235     \global\let\bbl@renewlist\@empty
3236     % Ends last section. See \bbl@inisec
3237     \def\bbl@elt##1##2{\bbl@inireader##1=##2\@@}%
3238     \bbl@cs{renew@\bbl@section}%
3239     \global\bbl@csarg\let{renew@\bbl@section}\relax
3240     \bbl@cs{secpost@\bbl@section}%
3241     \bbl@csarg{\global\expandafter\let}{inidata@\languagename}\bbl@inidata
3242     \bbl@exp{\\\bbl@add@list\\\bbl@ini@loaded{\languagename}}%
3243     \bbl@toglobal\bbl@ini@loaded
3244   \fi}
3245 \def\bbl@iniline#1\bbl@iniline{%
3246   \@ifnextchar[\bbl@inisec{\@ifnextchar;\bbl@iniskip\bbl@inipreread}#1\@@}% ]
```

The special cases for comment lines and sections are handled by the two following commands. In sections, we provide the posibility to take extra actions at the end or at the start (TODO - but note the last section is not ended). By default, key=val pairs are ignored. The `secpost` "hook" is used only by 'identification', while `secpre` only by `date.gregorian.licr`.

```
3247 \def\bbl@iniskip#1\@@{}%          if starts with ;
3248 \def\bbl@inisec[#1]#2\@@{%        if starts with opening bracket
3249   \def\bbl@elt##1##2{%
3250     \expandafter\toks@\expandafter{%
3251       \expandafter{\bbl@section}{##1}{##2}}%
3252     \bbl@exp{%
3253       \\\g@addto@macro\\\bbl@inidata{\\\bbl@elt\the\toks@}}%
3254     \bbl@inireader##1=##2\@@}%
3255   \bbl@cs{renew@\bbl@section}%
3256   \global\bbl@csarg\let{renew@\bbl@section}\relax
3257   \bbl@cs{secpost@\bbl@section}%
3258   % The previous code belongs to the previous section.
3259   % Now start the current one.
3260   \def\bbl@section{#1}%
3261   \def\bbl@elt##1##2{%
3262     \@namedef{bbl@KVP@#1/##1}{}}%
3263   \bbl@cs{renew@#1}%
3264   \bbl@cs{secpre@#1}%  pre-section `hook'
3265   \bbl@ifunset{bbl@inikv@#1}%
3266     {\let\bbl@inireader\bbl@iniskip}%
3267     {\bbl@exp{\let\\\bbl@inireader\<bbl@inikv@#1>}}}
3268 \let\bbl@renewlist\@empty
3269 \def\bbl@renewinikey#1/#2\@@#3{%
3270   \bbl@ifunset{bbl@renew@#1}%
3271     {\bbl@add@list\bbl@renewlist{#1}}%
3272     {}%
3273   \bbl@csarg\bbl@add{renew@#1}{\bbl@elt{#2}{#3}}}
```

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

```
3274 \def\bbl@inikv#1=#2\@@{%      key=value
3275   \bbl@trim@def\bbl@tempa{#1}%
3276   \bbl@trim\toks@{#2}%
3277   \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.

```
3278 \def\bbl@exportkey#1#2#3{%
3279   \bbl@ifunset{bbl@@kv@#2}%
3280     {\bbl@csarg\gdef{#1@\languagename}{#3}}%
3281     {\expandafter\ifx\csname bbl@@kv@#2\endcsname\@empty
3282       \bbl@csarg\gdef{#1@\languagename}{#3}%
```

```
3283        \else
3284          \bbl@exp{\global\let\<bbl@#1@\languagename>\<bbl@@kv@#2>}%
3285        \fi}}
```

Key-value pairs are treated differently depending on the section in the ini file. The following macros are the readers for identification and typography. Note \bbl@secpost@identification is called always (via \bbl@inisec), while \bbl@after@ini must be called explicitly after \bbl@read@ini if necessary.

```
3286 \def\bbl@iniwarning#1{%
3287    \bbl@ifunset{bbl@@kv@identification.warning#1}{}%
3288      {\bbl@warning{%
3289          From babel-\bbl@cs{lini@\languagename}.ini:\\%
3290          \bbl@cs{@kv@identification.warning#1}\\%
3291          Reported }}}
3292 \let\bbl@inikv@identification\bbl@inikv
3293 \def\bbl@secpost@identification{%
3294    \bbl@iniwarning{}%
3295    \ifcase\bbl@engine
3296      \bbl@iniwarning{.pdflatex}%
3297    \or
3298      \bbl@iniwarning{.lualatex}%
3299    \or
3300      \bbl@iniwarning{.xelatex}%
3301    \fi%
3302    \bbl@exportkey{elname}{identification.name.english}{}%
3303    \bbl@exp{\\\bbl@exportkey{lname}{identification.name.opentype}%
3304      {\csname bbl@elname@\languagename\endcsname}}%
3305    \bbl@exportkey{lbcp}{identification.tag.bcp47}{}%
3306    \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
3307    \bbl@exportkey{esname}{identification.script.name}{}%
3308    \bbl@exp{\\\bbl@exportkey{sname}{identification.script.name.opentype}%
3309      {\csname bbl@esname@\languagename\endcsname}}%
3310    \bbl@exportkey{sbcp}{identification.script.tag.bcp47}{}%
3311    \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}}
3312 \let\bbl@inikv@typography\bbl@inikv
3313 \let\bbl@inikv@characters\bbl@inikv
3314 \let\bbl@inikv@numbers\bbl@inikv
3315 \def\bbl@inikv@counters#1=#2\@@{%
3316    \def\bbl@tempc{#1}%
3317    \bbl@trim@def{\bbl@tempb*}{#2}%
3318    \in@{.1$}{#1$}%
3319    \ifin@
3320      \bbl@replace\bbl@tempc{.1}{}%
3321      \bbl@csarg\protected@xdef{cntr@\bbl@tempc @\languagename}{%
3322        \noexpand\bbl@alphnumeral{\bbl@tempc}}%
3323    \fi
3324    \in@{.F.}{#1}%
3325    \ifin@\else\in@{.S.}{#1}\fi
3326    \ifin@
3327      \bbl@csarg\protected@xdef{cntr@#1@\languagename}{\bbl@tempb*}%
3328    \else
3329      \toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa
3330      \expandafter\bbl@buildifcase\bbl@tempb* \\ % Space after \\
3331      \bbl@csarg{\global\expandafter\let}{cntr@#1@\languagename}\bbl@tempa
3332    \fi}
3333 \def\bbl@after@ini{%
3334    \bbl@exportkey{lfthm}{typography.lefthyphenmin}{2}%
3335    \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
3336    \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
```

```
3337  \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
3338  \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
3339  \bbl@exportkey{hyoth}{typography.hyphenate.other}{}%
3340  \bbl@exportkey{intsp}{typography.intraspace}{}%
3341  \bbl@exportkey{jstfy}{typography.justify}{w}%
3342  \bbl@exportkey{chrng}{characters.ranges}{}%
3343  \bbl@exportkey{dgnat}{numbers.digits.native}{}%
3344  \bbl@exportkey{rqtex}{identification.require.babel}{}%
3345  \bbl@toglobal\bbl@savetoday
3346  \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.

```
3347 \ifcase\bbl@engine
3348   \bbl@csarg\def{inikv@captions.licr}#1=#2\@@{%
3349     \bbl@ini@captions@aux{#1}{#2}}
3350 \else
3351   \def\bbl@inikv@captions#1=#2\@@{%
3352     \bbl@ini@captions@aux{#1}{#2}}
3353 \fi
```

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

```
3354 \def\bbl@ini@captions@aux#1#2{%
3355   \bbl@trim@def\bbl@tempa{#1}%
3356   \bbl@ifblank{#2}%
3357     {\bbl@exp{%
3358        \toks@{\\\bbl@nocaption{\bbl@tempa}{\languagename\bbl@tempa name}}}}%
3359     {\bbl@trim\toks@{#2}}%
3360   \bbl@exp{%
3361     \\\bbl@add\\\bbl@savestrings{%
3362       \\\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.

```
3363 \bbl@csarg\def{inikv@date.gregorian}#1=#2\@@{%        for defaults
3364   \bbl@inidate#1...\relax{#2}{}}
3365 \bbl@csarg\def{inikv@date.islamic}#1=#2\@@{%
3366   \bbl@inidate#1...\relax{#2}{islamic}}
3367 \bbl@csarg\def{inikv@date.hebrew}#1=#2\@@{%
3368   \bbl@inidate#1...\relax{#2}{hebrew}}
3369 \bbl@csarg\def{inikv@date.persian}#1=#2\@@{%
3370   \bbl@inidate#1...\relax{#2}{persian}}
3371 \bbl@csarg\def{inikv@date.indian}#1=#2\@@{%
3372   \bbl@inidate#1...\relax{#2}{indian}}
3373 \ifcase\bbl@engine
3374   \bbl@csarg\def{inikv@date.gregorian.licr}#1=#2\@@{%  override
3375     \bbl@inidate#1...\relax{#2}{}}
3376   \bbl@csarg\def{secpre@date.gregorian.licr}{%          discard uni
3377     \ifcase\bbl@engine\let\bbl@savedate\@empty\fi}
3378 \fi
3379 % eg: 1=months, 2=wide, 3=1, 4=dummy
3380 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
3381   \bbl@trim@def\bbl@tempa{#1.#2}%
3382   \bbl@ifsamestring{\bbl@tempa}{months.wide}%        to savedate
3383     {\bbl@trim@def\bbl@tempa{#3}%
3384      \bbl@trim\toks@{#5}%
```

139

```
3385        \bbl@exp{%
3386         \\\bbl@add\\\bbl@savedate{%
3387          \\\SetString\<month\romannumeral\bbl@tempa#6name>{\the\toks@}}}}%
3388     {\bbl@ifsamestring{\bbl@tempa}{date.long}%    defined now
3389       {\bbl@trim@def\bbl@toreplace{#5}%
3390        \bbl@TG@@date
3391        \global\bbl@csarg\let{date@\languagename}\bbl@toreplace
3392        \bbl@exp{%
3393          \gdef\<\languagename date>{\\\protect\<\languagename date >}%
3394          \gdef\<\languagename date >####1####2####3{%
3395            \\\bbl@usedategrouptrue
3396            \<bbl@ensure@\languagename>{%
3397              \<bbl@date@\languagename>{####1}{####2}{####3}}%
3398          \\\bbl@add\\\bbl@savetoday{%
3399            \\\SetString\\\today{%
3400              \<\languagename date>{\\\the\year}{\\\the\month}{\\\the\day}}}}}}%
3401        {}}
```

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.

```
3402 \let\bbl@calendar\@empty
3403 \newcommand\BabelDateSpace{\nobreakspace}
3404 \newcommand\BabelDateDot{.\@}
3405 \newcommand\BabelDated[1]{{\number#1}}
3406 \newcommand\BabelDatedd[1]{{\ifnum#1<10 0\fi\number#1}}
3407 \newcommand\BabelDateM[1]{{\number#1}}
3408 \newcommand\BabelDateMM[1]{{\ifnum#1<10 0\fi\number#1}}
3409 \newcommand\BabelDateMMMM[1]{{%
3410   \csname month\romannumeral#1\bbl@calendar name\endcsname}}%
3411 \newcommand\BabelDatey[1]{{\number#1}}%
3412 \newcommand\BabelDateyy[1]{{%
3413   \ifnum#1<10 0\number#1 %
3414   \else\ifnum#1<100 \number#1 %
3415   \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3416   \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
3417   \else
3418     \bbl@error
3419       {Currently two-digit years are restricted to the\\
3420        range 0-9999.}%
3421       {There is little you can do. Sorry.}%
3422   \fi\fi\fi\fi}}
3423 \newcommand\BabelDateyyyy[1]{{\number#1}} % FIXME - add leading 0
3424 \def\bbl@replace@finish@iii#1{%
3425   \bbl@exp{\def\\#1####1####2####3{\the\toks@}}}
3426 \def\bbl@TG@@date{%
3427   \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace{}}%
3428   \bbl@replace\bbl@toreplace{[.]}{\BabelDateDot{}}%
3429   \bbl@replace\bbl@toreplace{[d]}{\BabelDated{####3}}%
3430   \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{####3}}%
3431   \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{####2}}%
3432   \bbl@replace\bbl@toreplace{[MM]}{\BabelDateMM{####2}}%
3433   \bbl@replace\bbl@toreplace{[MMMM]}{\BabelDateMMMM{####2}}%
3434   \bbl@replace\bbl@toreplace{[y]}{\BabelDatey{####1}}%
3435   \bbl@replace\bbl@toreplace{[yy]}{\BabelDateyy{####1}}%
3436   \bbl@replace\bbl@toreplace{[yyyy]}{\BabelDateyyyy{####1}}%
3437 % Note after \bbl@replace \toks@ contains the resulting string.
3438 % TODO - Using this implicit behavior doesn't seem a good idea.
3439   \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.

```
3440 \def\bbl@provide@lsys#1{%
3441   \bbl@ifunset{bbl@lname@#1}%
3442     {\bbl@ini@basic{#1}}%
3443     {}%
3444   \bbl@csarg\let{lsys@#1}\@empty
3445   \bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}{}%
3446   \bbl@ifunset{bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{}%
3447   \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
3448   \bbl@ifunset{bbl@lname@#1}{}%
3449     {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
3450   \ifcase\bbl@engine\or\or
3451     \bbl@ifunset{bbl@prehc@#1}{}%
3452       {\bbl@exp{\\\bbl@ifblank{\bbl@cs{prehc@#1}}}%
3453         {}%
3454         {\bbl@csarg\bbl@add@list{lsys@#1}{HyphenChar="200B}}}%
3455   \fi
3456   \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 (which means any code in it must be skipped, too.

```
3457 \def\bbl@ini@basic#1{%
3458   \def\BabelBeforeIni##1##2{%
3459     \begingroup
3460       \bbl@add\bbl@secpost@identification{\closein\bbl@readstream }%
3461       \catcode`\[=12 \catcode`\]=12 \catcode`\==12 \catcode`\;=12 %
3462       \bbl@read@ini{##1}{font and identification data}%
3463       \endinput        % babel- .tex may contain onlypreamble's
3464     \endgroup}%           boxed, to avoid extra spaces:
3465   {\setbox\z@\hbox{\InputIfFileExists{babel-#1.tex}{}{}}}}
```

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

```
3466 \def\bbl@buildifcase#1 {% Returns \bbl@tempa, requires \toks@={}
3467   \ifx\\#1%              % \\ before, in case #1 is multiletter
3468     \bbl@exp{%
3469       \def\\\bbl@tempa####1{%
3470         \<ifcase>####1\space\the\toks@\<else>\\\@ctrerr\<fi>}}%
3471   \else
3472     \toks@\expandafter{\the\toks@\or #1}%
3473     \expandafter\bbl@buildifcase
3474   \fi}
```

The code for additive counters is somewhat tricky and it's based on the fact the arguments just before \@@ collects digits which have been left 'unused' in previous arguments, the first of them being the number of digits in the number to be converted. This explains the reverse set 76543210. Digits above 10000 are not handled yet. When the key contains the subkey `.F.`, the number after is treated as an special case. for a fixed form (see `babel-he.ini`, for example).

```
3475 \newcommand\localenumeral[2]{\bbl@cs{cntr@#1@\languagename}{#2}}
3476 \def\bbl@localecntr#1#2{\localenumeral{#2}{#1}}
3477 \newcommand\localecounter[2]{%
3478   \expandafter\bbl@localecntr\csname c@#2\endcsname{#1}}
3479 \def\bbl@alphnumeral#1#2{%
3480   \expandafter\bbl@alphnumeral@i\number#2 76543210\@@{#1}}
```

```
3481 \def\bbl@alphnumeral@i#1#2#3#4#5#6#7#8\@@#9{%
3482   \ifcase\@car#8\@nil\or   % Currenty <10000, but prepared for bigger
3483     \bbl@alphnumeral@ii{#9}000000#1\or
3484     \bbl@alphnumeral@ii{#9}00000#1#2\or
3485     \bbl@alphnumeral@ii{#9}0000#1#2#3\or
3486     \bbl@alphnumeral@ii{#9}000#1#2#3#4\else
3487     \bbl@alphnum@invalid{>9999}%
3488   \fi}
3489 \def\bbl@alphnumeral@ii#1#2#3#4#5#6#7#8{%
3490   \bbl@ifunset{bbl@cntr@#1.F.\number#5#6#7#8@\languagename}%
3491     {\bbl@cs{cntr@#1.4@\languagename}#5%
3492      \bbl@cs{cntr@#1.3@\languagename}#6%
3493      \bbl@cs{cntr@#1.2@\languagename}#7%
3494      \bbl@cs{cntr@#1.1@\languagename}#8%
3495      \ifnum#6#7#8>\z@ % An ad hod rule for Greek. Ugly. To be fixed.
3496        \bbl@ifunset{bbl@cntr@#1.S.321@\languagename}{}%
3497          {\bbl@cs{cntr@#1.S.321@\languagename}}%
3498      \fi}%
3499     {\bbl@cs{cntr@#1.F.\number#5#6#7#8@\languagename}}}
3500 \def\bbl@alphnum@invalid#1{%
3501   \bbl@error{Alphabetic numeral too large (#1)}%
3502     {Currently this is the limit.}}
```

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

```
3503 \newcommand\localeinfo[1]{%
3504   \bbl@ifunset{bbl@\csname bbl@info@#1\endcsname @\languagename}%
3505     {\bbl@error{I've found no info for the current locale.\\%
3506                 The corresponding ini file has not been loaded\\%
3507                 Perhaps it doesn't exist}%
3508                {See the manual for details.}}%
3509     {\bbl@cs{\csname bbl@info@#1\endcsname @\languagename}}}
3510 % \@namedef{bbl@info@name.locale}{lcname}
3511 \@namedef{bbl@info@tag.ini}{lini}
3512 \@namedef{bbl@info@name.english}{elname}
3513 \@namedef{bbl@info@name.opentype}{lname}
3514 \@namedef{bbl@info@tag.bcp47}{lbcp}
3515 \@namedef{bbl@info@tag.opentype}{lotf}
3516 \@namedef{bbl@info@script.name}{esname}
3517 \@namedef{bbl@info@script.name.opentype}{sname}
3518 \@namedef{bbl@info@script.tag.bcp47}{sbcp}
3519 \@namedef{bbl@info@script.tag.opentype}{sotf}
3520 \let\bbl@ensureinfo\@gobble
3521 \newcommand\BabelEnsureInfo{%
3522   \def\bbl@ensureinfo##1{%
3523     \ifx\InputIfFileExists\@undefined\else  % not in plain
3524       \bbl@ifunset{bbl@lname@##1}{\bbl@ini@basic{##1}}{}%
3525     \fi}}
```

More general, but non-expandable, is \getlocaleproperty. To inspect every possible loaded ini, we define \LocaleForEach, where \bbl@ini@loaded is a comma-separated list of locales, built by \bbl@read@ini.

```
3526 \newcommand\getlocaleproperty[3]{%
3527   \let#1\relax
3528   \def\bbl@elt##1##2##3{%
3529     \bbl@ifsamestring{##1/##2}{#3}%
3530       {\providecommand#1{##3}%
3531        \def\bbl@elt####1####2####3{}}%
3532       {}}%
```

142

```
3533  \bbl@cs{inidata@#2}%
3534  \ifx#1\relax
3535    \bbl@error
3536      {Unknown key for locale '#2':\\%
3537       #3\\%
3538       \string#1 will be set to \relax}%
3539      {Perhaps you misspelled it.}%
3540  \fi}
3541 \newcommand\LocaleForEach{\bbl@foreach\bbl@ini@loaded}
```

10   Adjusting the Babel bahavior

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

```
3542 \newcommand\babeladjust[1]{%  TODO. Error handling.
3543   \bbl@forkv{#1}{%
3544     \bbl@ifunset{bbl@ADJ@##1@##2}%
3545       {\bbl@cs{ADJ@##1}{##2}}%
3546       {\bbl@cs{ADJ@##1@##2}}}}
3547 %
3548 \def\bbl@adjust@lua#1#2{%
3549   \ifvmode
3550     \ifnum\currentgrouplevel=\z@
3551       \directlua{ Babel.#2 }%
3552       \expandafter\expandafter\expandafter\@gobble
3553     \fi
3554   \fi
3555   {\bbl@error   % The error is gobbled if everything went ok.
3556     {Currently, #1 related features can be adjusted only\\%
3557      in the main vertical list.}%
3558     {Maybe things change in the future, but this is what it is.}}}
3559 \@namedef{bbl@ADJ@bidi.mirroring@on}{%
3560   \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
3561 \@namedef{bbl@ADJ@bidi.mirroring@off}{%
3562   \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
3563 \@namedef{bbl@ADJ@bidi.text@on}{%
3564   \bbl@adjust@lua{bidi}{bidi_enabled=true}}
3565 \@namedef{bbl@ADJ@bidi.text@off}{%
3566   \bbl@adjust@lua{bidi}{bidi_enabled=false}}
3567 \@namedef{bbl@ADJ@bidi.mapdigits@on}{%
3568   \bbl@adjust@lua{bidi}{digits_mapped=true}}
3569 \@namedef{bbl@ADJ@bidi.mapdigits@off}{%
3570   \bbl@adjust@lua{bidi}{digits_mapped=false}}
3571 %
3572 \@namedef{bbl@ADJ@linebreak.sea@on}{%
3573   \bbl@adjust@lua{linebreak}{sea_enabled=true}}
3574 \@namedef{bbl@ADJ@linebreak.sea@off}{%
3575   \bbl@adjust@lua{linebreak}{sea_enabled=false}}
3576 \@namedef{bbl@ADJ@linebreak.cjk@on}{%
3577   \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
3578 \@namedef{bbl@ADJ@linebreak.cjk@off}{%
3579   \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
3580 %
3581 \def\bbl@adjust@layout#1{%
3582   \ifvmode
3583     #1%
3584     \expandafter\@gobble
3585   \fi
```

```
3586  {\bbl@error   % The error is gobbled if everything went ok.
3587     {Currently, layout related features can be adjusted only\\%
3588      in vertical mode.}%
3589     {Maybe things change in the future, but this is what it is.}}}
3590 \@namedef{bbl@ADJ@layout.tabular@on}{%
3591   \bbl@adjust@layout{\let\@tabular\bbl@NL@@tabular}}
3592 \@namedef{bbl@ADJ@layout.tabular@off}{%
3593   \bbl@adjust@layout{\let\@tabular\bbl@OL@@tabular}}
3594 \@namedef{bbl@ADJ@layout.lists@on}{%
3595   \bbl@adjust@layout{\let\list\bbl@NL@list}}
3596 \@namedef{bbl@ADJ@layout.lists@on}{%
3597   \bbl@adjust@layout{\let\list\bbl@OL@list}}
3598 \@namedef{bbl@ADJ@hyphenation.extra@on}{%
3599   \bbl@activateposthyphen}
3600 %
3601 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
3602   \bbl@bcpallowedtrue}
3603 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
3604   \bbl@bcpallowedfalse}
3605 \@namedef{bbl@ADJ@autoload.bcp47.prefix}#1{%
3606   \def\bbl@bcp@prefix{#1}}
3607 \def\bbl@bcp@prefix{bcp47-}
3608 \@namedef{bbl@ADJ@autoload.options}#1{%
3609   \def\bbl@autoload@options{#1}}
3610 \let\bbl@autoload@bcpoptions\@empty
3611 \@namedef{bbl@ADJ@autoload.bcp47.options}#1{%
3612   \def\bbl@autoload@bcpoptions{#1}}
3613 % TODO: use babel name, override
3614 %
3615 % As the final task, load the code for lua.
3616 %
3617 \ifx\directlua\@undefined\else
3618   \ifx\bbl@luapatterns\@undefined
3619     \input luababel.def
3620   \fi
3621 \fi
3622 ⟨/core⟩
```

A proxy file for switch.def

```
3623 ⟨*kernel⟩
3624 \let\bbl@onlyswitch\@empty
3625 \input babel.def
3626 \let\bbl@onlyswitch\@undefined
3627 ⟨/kernel⟩
3628 ⟨*patterns⟩
```

11   Loading hyphenation patterns

The following code is meant to be read by iniTeX because it should instruct TeX to read hyphenation patterns. To this end the docstrip option patterns can be used to include this code in the file hyphen.cfg. Code is written with lower level macros.
To make sure that LaTeX 2.09 executes the \@begindocumenthook we would want to alter \begin{document}, but as this done too often already, we add the new code at the front of \@preamblecmds. But we can only do that after it has been defined, so we add this piece of code to \dump.
This new definition starts by adding an instruction to write a message on the terminal and in the transcript file to inform the user of the preloaded hyphenation patterns.

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

```
3629 ⟨⟨Make sure ProvidesFile is defined⟩⟩
3630 \ProvidesFile{hyphen.cfg}[⟨⟨date⟩⟩ ⟨⟨version⟩⟩ Babel hyphens]
3631 \xdef\bbl@format{\jobname}
3632 \def\bbl@version{⟨⟨version⟩⟩}
3633 \def\bbl@date{⟨⟨date⟩⟩}
3634 \ifx\AtBeginDocument\@undefined
3635   \def\@empty{}
3636   \let\orig@dump\dump
3637   \def\dump{%
3638     \ifx\@ztryfc\@undefined
3639     \else
3640       \toks0=\expandafter{\@preamblecmds}%
3641       \edef\@preamblecmds{\noexpand\@begindocumenthook\the\toks0}%
3642       \def\@begindocumenthook{}%
3643     \fi
3644     \let\dump\orig@dump\let\orig@dump\@undefined\dump}
3645 \fi
3646 ⟨⟨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.

```
3647 \def\process@line#1#2 #3 #4 {%
3648   \ifx=#1%
3649     \process@synonym{#2}%
3650   \else
3651     \process@language{#1#2}{#3}{#4}%
3652   \fi
3653   \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.

```
3654 \toks@{}
3655 \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.

```
3656 \def\process@synonym#1{%
3657   \ifnum\last@language=\m@ne
3658     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
3659   \else
3660     \expandafter\chardef\csname l@#1\endcsname\last@language
3661     \wlog{\string\l@#1=\string\language\the\last@language}%
3662     \expandafter\let\csname #1hyphenmins\expandafter\endcsname
3663       \csname\languagename hyphenmins\endcsname
3664     \let\bbl@elt\relax
3665     \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}{}{}}%
3666   \fi}
```

\process@language The macro \process@language is used to process a non-empty line from the 'configuration file'. It has three arguments, each delimited by white space. The first argument is the

'name' of a language; the second is the name of the file that contains the patterns. The optional third argument is the name of a file containing hyphenation exceptions.

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

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

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

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

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

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

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

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

```
3667 \def\process@language#1#2#3{%
3668   \expandafter\addlanguage\csname l@#1\endcsname
3669   \expandafter\language\csname l@#1\endcsname
3670   \edef\languagename{#1}%
3671   \bbl@hook@everylanguage{#1}%
3672   %  > luatex
3673   \bbl@get@enc#1::\@@@
3674   \begingroup
3675     \lefthyphenmin\m@ne
3676     \bbl@hook@loadpatterns{#2}%
3677     %  > luatex
3678     \ifnum\lefthyphenmin=\m@ne
3679     \else
3680       \expandafter\xdef\csname #1hyphenmins\endcsname{%
3681         \the\lefthyphenmin\the\righthyphenmin}%
3682     \fi
3683   \endgroup
3684   \def\bbl@tempa{#3}%
3685   \ifx\bbl@tempa\@empty\else
3686     \bbl@hook@loadexceptions{#3}%
3687     %  > luatex
3688   \fi
3689   \let\bbl@elt\relax
3690   \edef\bbl@languages{%
3691     \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
3692   \ifnum\the\language=\z@
3693     \expandafter\ifx\csname #1hyphenmins\endcsname\relax
3694       \set@hyphenmins\tw@\thr@@\relax
```

```
3695    \else
3696      \expandafter\expandafter\expandafter\set@hyphenmins
3697        \csname #1hyphenmins\endcsname
3698    \fi
3699    \the\toks@
3700    \toks@{}%
3701  \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.

```
3702 \def\bbl@get@enc#1:#2:#3\@@@{\def\bbl@hyph@enc{#2}}
```

Now, hooks are defined. For efficiency reasons, they are dealt here in a special way.
Besides luatex, format-specific configuration files are taken into account. loadkernel
currently loads nothing, but define some basic macros instead.

```
3703 \def\bbl@hook@everylanguage#1{}
3704 \def\bbl@hook@loadpatterns#1{\input #1\relax}
3705 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
3706 \def\bbl@hook@loadkernel#1{%
3707  \def\addlanguage{\alloc@9\language\chardef\@cclvi}%
3708  \def\adddialect##1##2{%
3709    \global\chardef##1##2\relax
3710    \wlog{\string##1 = a dialect from \string\language##2}}%
3711  \def\iflanguage##1{%
3712    \expandafter\ifx\csname l@##1\endcsname\relax
3713      \@nolanerr{##1}%
3714    \else
3715      \ifnum\csname l@##1\endcsname=\language
3716        \expandafter\expandafter\expandafter\@firstoftwo
3717      \else
3718        \expandafter\expandafter\expandafter\@secondoftwo
3719      \fi
3720    \fi}%
3721  \def\providehyphenmins##1##2{%
3722    \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
3723      \@namedef{##1hyphenmins}{##2}%
3724    \fi}%
3725  \def\set@hyphenmins##1##2{%
3726    \lefthyphenmin##1\relax
3727    \righthyphenmin##2\relax}%
3728  \def\selectlanguage{%
3729    \errhelp{Selecting a language requires a package supporting it}%
3730    \errmessage{Not loaded}}%
3731  \let\foreignlanguage\selectlanguage
3732  \let\otherlanguage\selectlanguage
3733  \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
3734  \def\setlocale{%
3735    \errhelp{Find an armchair, sit down and wait}%
3736    \errmessage{Not yet available}}%
3737  \let\uselocale\setlocale
3738  \let\locale\setlocale
3739  \let\selectlocale\setlocale
3740  \let\localename\setlocale
3741  \let\textlocale\setlocale
3742  \let\textlanguage\setlocale
3743  \let\languagetext\setlocale}
3744 \begingroup
3745  \def\AddBabelHook#1#2{%
```

147

```
3746     \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
3747       \def\next{\toks1}%
3748     \else
3749       \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname####1}%
3750     \fi
3751     \next}
3752   \ifx\directlua\@undefined
3753     \ifx\XeTeXinputencoding\@undefined\else
3754       \input xebabel.def
3755     \fi
3756   \else
3757     \input luababel.def
3758   \fi
3759   \openin1 = babel-\bbl@format.cfg
3760   \ifeof1
3761   \else
3762     \input babel-\bbl@format.cfg\relax
3763   \fi
3764   \closein1
3765 \endgroup
3766 \bbl@hook@loadkernel{switch.def}
```

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

```
3767 \openin1 = language.dat
```

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

```
3768 \def\languagename{english}%
3769 \ifeof1
3770   \message{I couldn't find the file language.dat,\space
3771           I will try the file hyphen.tex}
3772   \input hyphen.tex\relax
3773   \chardef\l@english\z@
3774 \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.

```
3775   \last@language\m@ne
```

We now read lines from the file until the end is found. While reading from the input, it is useful to switch off recognition of the end-of-line character. This saves us stripping off spaces from the contents of the control sequence.

```
3776   \loop
3777     \endlinechar\m@ne
3778     \read1 to \bbl@line
3779     \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.

```
3780     \if T\ifeof1F\fi T\relax
3781       \ifx\bbl@line\@empty\else
3782         \edef\bbl@line{\bbl@line\space\space\space}%
3783         \expandafter\process@line\bbl@line\relax
3784       \fi
3785   \repeat
```

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

```
3786  \begingroup
3787    \def\bbl@elt#1#2#3#4{%
3788      \global\language=#2\relax
3789      \gdef\languagename{#1}%
3790      \def\bbl@elt##1##2##3##4{}}%
3791    \bbl@languages
3792  \endgroup
3793 \fi
3794 \closein1
```

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

```
3795 \if/\the\toks@/\else
3796   \errhelp{language.dat loads no language, only synonyms}
3797   \errmessage{Orphan language synonym}
3798 \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.

```
3799 \let\bbl@line\@undefined
3800 \let\process@line\@undefined
3801 \let\process@synonym\@undefined
3802 \let\process@language\@undefined
3803 \let\bbl@get@enc\@undefined
3804 \let\bbl@hyph@enc\@undefined
3805 \let\bbl@tempa\@undefined
3806 \let\bbl@hook@loadkernel\@undefined
3807 \let\bbl@hook@everylanguage\@undefined
3808 \let\bbl@hook@loadpatterns\@undefined
3809 \let\bbl@hook@loadexceptions\@undefined
3810 ⟨/patterns⟩
```

Here the code for iniTeX ends.

12 Font handling with fontspec

Add the bidi handler just before luaoftload, which is loaded by default by LaTeX. Just in
case, consider the possibility it has not been loaded. First, a couple of definitions related to
bidi [misplaced].

```
3811 ⟨⟨*More package options⟩⟩ ≡
3812 \ifodd\bbl@engine
3813   \DeclareOption{bidi=basic-r}%
3814     {\ExecuteOptions{bidi=basic}}
3815   \DeclareOption{bidi=basic}%
3816     {\let\bbl@beforeforeign\leavevmode
3817      % TODO - to locale_props, not as separate attribute
3818      \newattribute\bbl@attr@dir
3819      % I don't like it, hackish:
3820      \frozen@everymath\expandafter{%
3821        \expandafter\bbl@mathboxdir\the\frozen@everymath}%
3822      \frozen@everydisplay\expandafter{%
3823        \expandafter\bbl@mathboxdir\the\frozen@everydisplay}%
3824      \bbl@exp{\output{\bodydir\pagedir\the\output}}%
3825      \AtEndOfPackage{\EnableBabelHook{babel-bidi}}}}
```

```
3826 \else
3827   \DeclareOption{bidi=basic-r}%
3828     {\ExecuteOptions{bidi=basic}}
3829   \DeclareOption{bidi=basic}%
3830     {\bbl@error
3831       {The bidi method `basic' is available only in\\%
3832        luatex. I'll continue with `bidi=default', so\\%
3833        expect wrong results}%
3834      {See the manual for further details.}%
3835     \let\bbl@beforeforeign\leavevmode
3836     \AtEndOfPackage{%
3837       \EnableBabelHook{babel-bidi}%
3838       \bbl@xebidipar}}
3839   \def\bbl@loadxebidi#1{%
3840     \ifx\RTLfootnotetext\@undefined
3841       \AtEndOfPackage{%
3842         \EnableBabelHook{babel-bidi}%
3843         \ifx\fontspec\@undefined
3844           \usepackage{fontspec}% bidi needs fontspec
3845         \fi
3846         \usepackage#1{bidi}}%
3847     \fi}
3848   \DeclareOption{bidi=bidi}%
3849     {\bbl@tentative{bidi=bidi}%
3850      \bbl@loadxebidi{}}
3851   \DeclareOption{bidi=bidi-r}%
3852     {\bbl@tentative{bidi=bidi-r}%
3853      \bbl@loadxebidi{[rldocument]}}
3854   \DeclareOption{bidi=bidi-l}%
3855     {\bbl@tentative{bidi=bidi-l}%
3856      \bbl@loadxebidi{}}
3857 \fi
3858 \DeclareOption{bidi=default}%
3859   {\let\bbl@beforeforeign\leavevmode
3860    \ifodd\bbl@engine
3861      \newattribute\bbl@attr@dir
3862      \bbl@exp{\output{\bodydir\pagedir\the\output}}}%
3863    \fi
3864    \AtEndOfPackage{%
3865      \EnableBabelHook{babel-bidi}%
3866      \ifodd\bbl@engine\else
3867        \bbl@xebidipar
3868      \fi}}
3869 ⟨⟨/More package options⟩⟩
```

With explicit languages, we could define the font at once, but we don't. Just wait and see if the language is actually activated. bbl@font replaces hardcoded font names inside \..family by the corresponding macro \..default.

```
3870 ⟨⟨∗Font selection⟩⟩ ≡
3871 \bbl@trace{Font handling with fontspec}
3872 \@onlypreamble\babelfont
3873 \newcommand\babelfont[2][]{%  1=langs/scripts 2=fam
3874   \bbl@foreach{#1}{%
3875     \expandafter\ifx\csname date##1\endcsname\relax
3876     \IfFileExists{babel-##1.tex}%
3877       {\babelprovide{##1}}%
3878       {}%
3879     \fi}%
3880   \edef\bbl@tempa{#1}%
```

```
3881    \def\bbl@tempb{#2}%  Used by \bbl@bblfont
3882    \ifx\fontspec\@undefined
3883      \usepackage{fontspec}%
3884    \fi
3885    \EnableBabelHook{babel-fontspec}% Just calls \bbl@switchfont
3886    \bbl@bblfont}
3887 \newcommand\bbl@bblfont[2][]{% 1=features 2=fontname, @font=rm|sf|tt
3888    \bbl@ifunset{\bbl@tempb family}%
3889      {\bbl@providefam{\bbl@tempb}}%
3890      {\bbl@exp{%
3891        \\\bbl@sreplace\<\bbl@tempb family >%
3892          {\@nameuse{\bbl@tempb default}}{\<\bbl@tempb default>}}}%
3893    % For the default font, just in case:
3894    \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
3895    \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
3896      {\bbl@csarg\edef{\bbl@tempb dflt@}{<>{#1}{#2}}% save bbl@rmdflt@
3897       \bbl@exp{%
3898         \let\<bbl@\bbl@tempb dflt@\languagename>\<bbl@\bbl@tempb dflt@>%
3899         \\\bbl@font@set\<bbl@\bbl@tempb dflt@\languagename>%
3900                        \<\bbl@tempb default>\<\bbl@tempb family>}}%
3901      {\bbl@foreach\bbl@tempa{% ie bbl@rmdflt@lang / *scrt
3902         \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:

```
3903 \def\bbl@providefam#1{%
3904    \bbl@exp{%
3905      \\\newcommand\<#1default>{}% Just define it
3906      \\\bbl@add@list\\\bbl@font@fams{#1}%
3907      \\\DeclareRobustCommand\<#1family>{%
3908        \\\not@math@alphabet\<#1family>\relax
3909        \\\fontfamily\<#1default>\\\selectfont}%
3910      \\\DeclareTextFontCommand{\<text#1>}{\<#1family>}}}
```

The following macro is activated when the hook `babel-fontspec` is enabled. But before we define a macro for a warning, which sets a flag to avoid duplicate them.

```
3911 \def\bbl@nostdfont#1{%
3912    \bbl@ifunset{bbl@WFF@\f@family}%
3913      {\bbl@csarg\gdef{WFF@\f@family}{}%  Flag, to avoid dupl warns
3914       \bbl@infowarn{The current font is not a babel standard family:\\%
3915         #1%
3916         \fontname\font\\%
3917         There is nothing intrinsically wrong with this warning, and\\%
3918         you can ignore it altogether if you do not need these\\%
3919         families. But if they are used in the document, you should be\\%
3920         aware 'babel' will no set Script and Language for them, so\\%
3921         you may consider defining a new family with \string\babelfont.\\%
3922         See the manual for further details about \string\babelfont.\\%
3923         Reported}}%
3924      {}}%
3925 \gdef\bbl@switchfont{%
3926    \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}%
3927    \bbl@exp{%  eg Arabic -> arabic
3928      \lowercase{\edef\\\bbl@tempa{\bbl@cl{sname}}}}%
3929    \bbl@foreach\bbl@font@fams{%
3930      \bbl@ifunset{bbl@##1dflt@\languagename}%      (1) language?
3931        {\bbl@ifunset{bbl@##1dflt@*\bbl@tempa}%     (2) from script?
3932          {\bbl@ifunset{bbl@##1dflt@}%              2=F - (3) from generic?
3933            {}%                                     123=F - nothing!
3934            {\bbl@exp{%                             3=T - from generic
```

```
3935              \global\let\<bbl@##1dflt@\languagename>%
3936                      \<bbl@##1dflt@>}}}%
3937         {\bbl@exp{%                          2=T - from script
3938            \global\let\<bbl@##1dflt@\languagename>%
3939                     \<bbl@##1dflt@*\bbl@tempa>}}}%
3940      {}}%                                1=T - language, already defined
3941 \def\bbl@tempa{\bbl@nostdfont{}}%
3942 \bbl@foreach\bbl@font@fams{%      don't gather with prev for
3943   \bbl@ifunset{bbl@##1dflt@\languagename}%
3944     {\bbl@cs{famrst@##1}%
3945      \global\bbl@csarg\let{famrst@##1}\relax}%
3946     {\bbl@exp{% order is relevant
3947       \\\bbl@add\\\originalTeX{%
3948         \\\bbl@font@rst{\bbl@cl{##1dflt}}%
3949                     \<##1default>\<##1family>{##1}}%
3950       \\\bbl@font@set\<bbl@##1dflt@\languagename>% the main part!
3951                   \<##1default>\<##1family>}}}%
3952 \bbl@ifrestoring{}{\bbl@tempa}}%
```

The following is executed at the beginning of the aux file or the document to warn about fonts not defined with \babelfont.

```
3953 \ifx\f@family\@undefined\else   % if latex
3954  \ifcase\bbl@engine             % if pdftex
3955    \let\bbl@ckeckstdfonts\relax
3956  \else
3957    \def\bbl@ckeckstdfonts{%
3958      \begingroup
3959        \global\let\bbl@ckeckstdfonts\relax
3960        \let\bbl@tempa\@empty
3961        \bbl@foreach\bbl@font@fams{%
3962          \bbl@ifunset{bbl@##1dflt@}%
3963            {\@nameuse{##1family}%
3964             \bbl@csarg\gdef{WFF@\f@family}{}% Flag
3965             \bbl@exp{\\\bbl@add\\\bbl@tempa{* \<##1family>= \f@family\\\\%
3966                \space\space\fontname\font\\\\}}%
3967             \bbl@csarg\xdef{##1dflt@}{\f@family}%
3968             \expandafter\xdef\csname ##1default\endcsname{\f@family}}%
3969            {}}%
3970        \ifx\bbl@tempa\@empty\else
3971          \bbl@infowarn{The following font families will use the default\\%
3972            settings for all or some languages:\\%
3973            \bbl@tempa
3974            There is nothing intrinsically wrong with it, but\\%
3975            'babel' will no set Script and Language, which could\\%
3976             be relevant in some languages. If your document uses\\%
3977             these families, consider redefining them with \string\babelfont.\\%
3978            Reported}%
3979        \fi
3980      \endgroup}
3981  \fi
3982 \fi
```

Now the macros defining the font with fontspec.
When there are repeated keys in fontspec, the last value wins. So, we just place the ini settings at the beginning, and user settings will take precedence. We must deactivate temporarily \bbl@mapselect because \selectfont is called internally when a font is defined.

```
3983 \def\bbl@font@set#1#2#3{% eg \bbl@rmdflt@lang \rmdefault \rmfamily
3984   \bbl@xin@{<>}{#1}%
```

```
3985    \ifin@
3986      \bbl@exp{\\\bbl@fontspec@set\\#1\expandafter\@gobbletwo#1\\#3}%
3987    \fi
3988    \bbl@exp{%
3989      \def\\#2{#1}%            eg, \rmdefault{\bbl@rmdflt@lang}
3990      \\\bbl@ifsamestring{#2}{\f@family}{\\#3\let\\\bbl@tempa\relax}{}}}
3991 %      TODO - next should be global?, but even local does its job. I'm
3992 %      still not sure -- must investigate:
3993 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
3994    \let\bbl@tempe\bbl@mapselect
3995    \let\bbl@mapselect\relax
3996    \let\bbl@temp@fam#4%        eg, '\rmfamily', to be restored below
3997    \let#4\@empty        %       Make sure \renewfontfamily is valid
3998    \bbl@exp{%
3999      \let\\\bbl@temp@pfam\<\bbl@stripslash#4\space>% eg, '\rmfamily '
4000      \<keys_if_exist:nnF>{fontspec-opentype}{Script/\bbl@cl{sname}}%
4001        {\\\newfontscript{\bbl@cl{sname}}{\bbl@cl{sotf}}}%
4002      \<keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@cl{lname}}%
4003        {\\\newfontlanguage{\bbl@cl{lname}}{\bbl@cl{lotf}}}%
4004      \\\renewfontfamily\\#4%
4005        [\bbl@cs{lsys@\languagename},#2]}{#3}% ie \bbl@exp{..}{#3}
4006    \begingroup
4007      #4%
4008      \xdef#1{\f@family}%      eg, \bbl@rmdflt@lang{FreeSerif(0)}
4009    \endgroup
4010    \let#4\bbl@temp@fam
4011    \bbl@exp{\let\<\bbl@stripslash#4\space>}\bbl@temp@pfam
4012    \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.

```
4013 \def\bbl@font@rst#1#2#3#4{%
4014    \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.

```
4015 \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 :-).

```
4016 \newcommand\babelFSstore[2][]{%
4017    \bbl@ifblank{#1}%
4018      {\bbl@csarg\def{sname@#2}{Latin}}%
4019      {\bbl@csarg\def{sname@#2}{#1}}%
4020    \bbl@provide@dirs{#2}%
4021    \bbl@csarg\ifnum{wdir@#2}>\z@
4022      \let\bbl@beforeforeign\leavevmode
4023      \EnableBabelHook{babel-bidi}%
4024    \fi
4025    \bbl@foreach{#2}{%
4026      \bbl@FSstore{##1}{rm}\rmdefault\bbl@save@rmdefault
4027      \bbl@FSstore{##1}{sf}\sfdefault\bbl@save@sfdefault
4028      \bbl@FSstore{##1}{tt}\ttdefault\bbl@save@ttdefault}}
4029 \def\bbl@FSstore#1#2#3#4{%
4030    \bbl@csarg\edef{#2default#1}{#3}%
4031    \expandafter\addto\csname extras#1\endcsname{%
4032      \let#4#3%
4033      \ifx#3\f@family
```

```
4034        \edef#3{\csname bbl@#2default#1\endcsname}%
4035        \fontfamily{#3}\selectfont
4036      \else
4037        \edef#3{\csname bbl@#2default#1\endcsname}%
4038      \fi}%
4039    \expandafter\addto\csname noextras#1\endcsname{%
4040      \ifx#3\f@family
4041        \fontfamily{#4}\selectfont
4042      \fi
4043      \let#3#4}}
4044  \let\bbl@langfeatures\@empty
4045  \def\babelFSfeatures{% make sure \fontspec is redefined once
4046    \let\bbl@ori@fontspec\fontspec
4047    \renewcommand\fontspec[1][]{%
4048      \bbl@ori@fontspec[\bbl@langfeatures##1]}
4049    \let\babelFSfeatures\bbl@FSfeatures
4050    \babelFSfeatures}
4051  \def\bbl@FSfeatures#1#2{%
4052    \expandafter\addto\csname extras#1\endcsname{%
4053      \babel@save\bbl@langfeatures
4054      \edef\bbl@langfeatures{#2,}}}
4055 ⟨⟨/Font selection⟩⟩
```

13 Hooks for XeTeX and LuaTeX

13.1 XeTeX

Unfortunately, the current encoding cannot be retrieved and therefore it is reset always to
utf8, which seems a sensible default.

```
4056 ⟨⟨*Footnote changes⟩⟩ ≡
4057 \bbl@trace{Bidi footnotes}
4058 \ifx\bbl@beforeforeign\leavevmode
4059   \def\bbl@footnote#1#2#3{%
4060     \@ifnextchar[%
4061       {\bbl@footnote@o{#1}{#2}{#3}}%
4062       {\bbl@footnote@x{#1}{#2}{#3}}}
4063   \def\bbl@footnote@x#1#2#3#4{%
4064     \bgroup
4065       \select@language@x{\bbl@main@language}%
4066       \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
4067     \egroup}
4068   \def\bbl@footnote@o#1#2#3[#4]#5{%
4069     \bgroup
4070       \select@language@x{\bbl@main@language}%
4071       \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
4072     \egroup}
4073   \def\bbl@footnotetext#1#2#3{%
4074     \@ifnextchar[%
4075       {\bbl@footnotetext@o{#1}{#2}{#3}}%
4076       {\bbl@footnotetext@x{#1}{#2}{#3}}}
4077   \def\bbl@footnotetext@x#1#2#3#4{%
4078     \bgroup
4079       \select@language@x{\bbl@main@language}%
4080       \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
4081     \egroup}
4082   \def\bbl@footnotetext@o#1#2#3[#4]#5{%
4083     \bgroup
```

154

```
4084        \select@language@x{\bbl@main@language}%
4085        \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
4086      \egroup}
4087    \def\BabelFootnote#1#2#3#4{%
4088      \ifx\bbl@fn@footnote\@undefined
4089        \let\bbl@fn@footnote\footnote
4090      \fi
4091      \ifx\bbl@fn@footnotetext\@undefined
4092        \let\bbl@fn@footnotetext\footnotetext
4093      \fi
4094      \bbl@ifblank{#2}%
4095        {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
4096         \@namedef{\bbl@stripslash#1text}%
4097           {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
4098        {\def#1{\bbl@exp{\\\bbl@footnote{\\\foreignlanguage{#2}}{#3}{#4}}%
4099         \@namedef{\bbl@stripslash#1text}%
4100           {\bbl@exp{\\\bbl@footnotetext{\\\foreignlanguage{#2}}{#3}{#4}}}}}
4101    \fi
4102    ⟨⟨/Footnote changes⟩⟩
```

Now, the code.

```
4103    ⟨∗xetex⟩
4104    \def\BabelStringsDefault{unicode}
4105    \let\xebbl@stop\relax
4106    \AddBabelHook{xetex}{encodedcommands}{%
4107      \def\bbl@tempa{#1}%
4108      \ifx\bbl@tempa\@empty
4109        \XeTeXinputencoding"bytes"%
4110      \else
4111        \XeTeXinputencoding"#1"%
4112      \fi
4113      \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4114    \AddBabelHook{xetex}{stopcommands}{%
4115      \xebbl@stop
4116      \let\xebbl@stop\relax}
4117    \def\bbl@intraspace#1 #2 #3\@@{%
4118      \bbl@csarg\gdef{xeisp@\languagename}%
4119        {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4120    \def\bbl@intrapenalty#1\@@{%
4121      \bbl@csarg\gdef{xeipn@\languagename}%
4122        {\XeTeXlinebreakpenalty #1\relax}}
4123    \def\bbl@provide@intraspace{%
4124      \bbl@xin@{\bbl@cl{lnbrk}}{s}%
4125      \ifin@\else\bbl@xin@{\bbl@cl{lnbrk}}{c}\fi
4126      \ifin@
4127        \bbl@ifunset{bbl@intsp@\languagename}{}%
4128          {\expandafter\ifx\csname bbl@intsp@\languagename\endcsname\@empty\else
4129            \ifx\bbl@KVP@intraspace\@nil
4130              \bbl@exp{%
4131                \\\bbl@intraspace\bbl@cl{intsp}\\\@@}%
4132            \fi
4133            \ifx\bbl@KVP@intrapenalty\@nil
4134              \bbl@intrapenalty0\@@
4135            \fi
4136          \fi
4137          \ifx\bbl@KVP@intraspace\@nil\else % We may override the ini
4138            \expandafter\bbl@intraspace\bbl@KVP@intraspace\@@
4139          \fi
4140          \ifx\bbl@KVP@intrapenalty\@nil\else
```

```
4141            \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
4142        \fi
4143        \bbl@exp{%
4144          \\\bbl@add\<extras\languagename>{%
4145            \XeTeXlinebreaklocale "\bbl@cl{lbcp}"%
4146            \<bbl@xeisp@\languagename>%
4147            \<bbl@xeipn@\languagename>}%
4148          \\\bbl@toglobal\<extras\languagename>%
4149          \\\bbl@add\<noextras\languagename>{%
4150            \XeTeXlinebreaklocale "en"}%
4151          \\\bbl@toglobal\<noextras\languagename>}%
4152        \ifx\bbl@ispacesize\@undefined
4153          \gdef\bbl@ispacesize{\bbl@cl{xeisp}}%
4154          \ifx\AtBeginDocument\@notprerr
4155            \expandafter\@secondoftwo  % to execute right now
4156          \fi
4157          \AtBeginDocument{%
4158            \expandafter\bbl@add
4159            \csname selectfont \endcsname{\bbl@ispacesize}%
4160            \expandafter\bbl@toglobal\csname selectfont \endcsname}%
4161        \fi}%
4162  \fi}
4163 \ifx\DisableBabelHook\@undefined\endinput\fi
4164 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4165 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
4166 \DisableBabelHook{babel-fontspec}
4167 ⟨⟨Font selection⟩⟩
4168 \input txtbabel.def
4169 ⟨/xetex⟩
```

13.2   Layout

In progress.

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

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

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

```
4170 ⟨∗texxet⟩
4171 \providecommand\bbl@provide@intraspace{}
4172 \bbl@trace{Redefinitions for bidi layout}
4173 \def\bbl@sspre@caption{%
4174   \bbl@exp{\everyhbox{\\\bbl@textdir\bbl@cs{wdir@\bbl@main@language}}}}
4175 \ifx\bbl@opt@layout\@nnil\endinput\fi  % No layout
4176 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
4177 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
4178 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
4179   \def\@hangfrom#1{%
4180     \setbox\@tempboxa\hbox{{#1}}%
4181     \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
4182     \noindent\box\@tempboxa}
4183   \def\raggedright{%
4184     \let\\\@centercr
4185     \bbl@startskip\z@skip
4186     \@rightskip\@flushglue
```

```
4187     \bbl@endskip\@rightskip
4188     \parindent\z@
4189     \parfillskip\bbl@startskip}
4190   \def\raggedleft{%
4191     \let\\\@centercr
4192     \bbl@startskip\@flushglue
4193     \bbl@endskip\z@skip
4194     \parindent\z@
4195     \parfillskip\bbl@endskip}
4196 \fi
4197 \IfBabelLayout{lists}
4198   {\bbl@sreplace\list
4199     {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
4200   \def\bbl@listleftmargin{%
4201     \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
4202   \ifcase\bbl@engine
4203     \def\labelenumii{)\theenumii(}% pdftex doesn't reverse ()
4204     \def\p@enumiii{\p@enumii)\theenumii(}%
4205   \fi
4206   \bbl@sreplace\@verbatim
4207     {\leftskip\@totalleftmargin}%
4208     {\bbl@startskip\textwidth
4209      \advance\bbl@startskip-\linewidth}%
4210   \bbl@sreplace\@verbatim
4211     {\rightskip\z@skip}%
4212     {\bbl@endskip\z@skip}}%
4213   {}
4214 \IfBabelLayout{contents}
4215   {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
4216    \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
4217   {}
4218 \IfBabelLayout{columns}
4219   {\bbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbl@outputhbox}%
4220   \def\bbl@outputhbox#1{%
4221     \hb@xt@\textwidth{%
4222       \hskip\columnwidth
4223       \hfil
4224       {\normalcolor\vrule \@width\columnseprule}%
4225       \hfil
4226       \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
4227       \hskip-\textwidth
4228       \hb@xt@\columnwidth{\box\@outputbox \hss}%
4229       \hskip\columnsep
4230       \hskip\columnwidth}}}%
4231   {}
4232 ⟨⟨Footnote changes⟩⟩
4233 \IfBabelLayout{footnotes}%
4234   {\BabelFootnote\footnote\languagename{}{}%
4235    \BabelFootnote\localfootnote\languagename{}{}%
4236    \BabelFootnote\mainfootnote{}{}{}}
4237   {}
```

Implicitly reverses sectioning labels in `bidi=basic`, because the full stop is not in contact with L numbers any more. I think there must be a better way.

```
4238 \IfBabelLayout{counters}%
4239   {\let\bbl@latinarabic=\@arabic
4240    \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
4241    \let\bbl@asciiroman=\@roman
4242    \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciiroman#1}}}%
```

```
4243      \let\bbl@asciiRoman=\@Roman
4244      \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}}}{}
4245 ⟨/texxet⟩
```

13.3 LuaTeX

The loader for luatex is based solely on `language.dat`, which is read on the fly. The code shouldn't be executed when the format is build, so we check if `\AddBabelHook` is defined. Then comes a modified version of the loader in `hyphen.cfg` (without the hyphenmins stuff, which is under the direct control of babel).

The names `\l@<language>` are defined and take some value from the beginning because all `ldf` files assume this for the corresponding language to be considered valid, but patterns are not loaded (except the first one). This is done later, when the language is first selected (which usually means when the `ldf` finishes). If a language has been loaded, `\bbl@hyphendata@<num>` exists (with the names of the files read).

The default setup preloads the first language into the format. This is intended mainly for 'english', so that it's available without further intervention from the user. To avoid duplicating it, the following rule applies: if the "0th" language and the first language in `language.dat` have the same name then just ignore the latter. If there are new synonymous, the are added, but note if the language patterns have not been preloaded they won't at run time.

Other preloaded languages could be read twice, if they have been preloaded into the format. This is not optimal, but it shouldn't happen very often – with luatex patterns are best loaded when the document is typeset, and the "0th" language is preloaded just for backwards compatibility.

As of 1.1b, lua(e)tex is taken into account. Formerly, loading of patterns on the fly didn't work in this format, but with the new loader it does. Unfortunately, the format is not based on babel, and data could be duplicated, because languages are reassigned above those in the format (nothing serious, anyway). Note even with this format `language.dat` is used (under the principle of a single source), instead of `language.def`.

Of course, there is room for improvements, like tools to read and reassign languages, which would require modifying the language list, and better error handling.

We need catcode tables, but no format (targeted by babel) provide a command to allocate them (although there are packages like ctablestack). FIX - This isn't true anymore. For the moment, a dangerous approach is used - just allocate a high random number and cross the fingers. To complicate things, etex.sty changes the way languages are allocated.

This files is read at three places: (1) when `plain.def`, `babel.sty` starts, to read the list of available languages from `language.dat` (for the `base` option); (2) at hyphen.cfg, to modify some macros; (3) in the middle of `plain.def` and `babel.sty`, by `babel.def`, with the commands and other definitions for luatex (eg, `\babelpatterns`).

```
4246 ⟨*luatex⟩
4247 \ifx\AddBabelHook\@undefined % When plain.def, babel.sty starts
4248 \bbl@trace{Read language.dat}
4249 \ifx\bbl@readstream\@undefined
4250   \csname newread\endcsname\bbl@readstream
4251 \fi
4252 \begingroup
4253   \toks@{}
4254   \count@\z@ % 0=start, 1=0th, 2=normal
4255   \def\bbl@process@line#1#2 #3 #4 {%
4256     \ifx=#1%
4257       \bbl@process@synonym{#2}%
4258     \else
4259       \bbl@process@language{#1#2}{#3}{#4}%
4260     \fi
```

```
4261     \ignorespaces}
4262 \def\bbl@manylang{%
4263     \ifnum\bbl@last>\@ne
4264       \bbl@info{Non-standard hyphenation setup}%
4265     \fi
4266     \let\bbl@manylang\relax}
4267 \def\bbl@process@language#1#2#3{%
4268     \ifcase\count@
4269       \@ifundefined{zth@#1}{\count@\tw@}{\count@\@ne}%
4270     \or
4271       \count@\tw@
4272     \fi
4273     \ifnum\count@=\tw@
4274       \expandafter\addlanguage\csname l@#1\endcsname
4275       \language\allocationnumber
4276       \chardef\bbl@last\allocationnumber
4277       \bbl@manylang
4278       \let\bbl@elt\relax
4279       \xdef\bbl@languages{%
4280         \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
4281     \fi
4282     \the\toks@
4283     \toks@{}}
4284 \def\bbl@process@synonym@aux#1#2{%
4285     \global\expandafter\chardef\csname l@#1\endcsname#2\relax
4286     \let\bbl@elt\relax
4287     \xdef\bbl@languages{%
4288       \bbl@languages\bbl@elt{#1}{#2}{}{}}}%
4289 \def\bbl@process@synonym#1{%
4290     \ifcase\count@
4291       \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
4292     \or
4293       \@ifundefined{zth@#1}{\bbl@process@synonym@aux{#1}{0}}{}%
4294     \else
4295       \bbl@process@synonym@aux{#1}{\the\bbl@last}%
4296     \fi}
4297 \ifx\bbl@languages\@undefined % Just a (sensible?) guess
4298     \chardef\l@english\z@
4299     \chardef\l@USenglish\z@
4300     \chardef\bbl@last\z@
4301     \global\@namedef{bbl@hyphendata@0}{{hyphen.tex}{}}
4302     \gdef\bbl@languages{%
4303       \bbl@elt{english}{0}{hyphen.tex}{}%
4304       \bbl@elt{USenglish}{0}{}{}}
4305 \else
4306     \global\let\bbl@languages@format\bbl@languages
4307     \def\bbl@elt#1#2#3#4{% Remove all except language 0
4308       \ifnum#2>\z@\else
4309         \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
4310       \fi}%
4311     \xdef\bbl@languages{\bbl@languages}%
4312 \fi
4313 \def\bbl@elt#1#2#3#4{\@namedef{zth@#1}{}} % Define flags
4314 \bbl@languages
4315 \openin\bbl@readstream=language.dat
4316 \ifeof\bbl@readstream
4317     \bbl@warning{I couldn't find language.dat. No additional\\%
4318                  patterns loaded. Reported}%
4319 \else
```

```
4320    \loop
4321      \endlinechar\m@ne
4322      \read\bbl@readstream to \bbl@line
4323      \endlinechar`\^^M
4324      \if T\ifeof\bbl@readstream F\fi T\relax
4325        \ifx\bbl@line\@empty\else
4326          \edef\bbl@line{\bbl@line\space\space\space}%
4327          \expandafter\bbl@process@line\bbl@line\relax
4328        \fi
4329      \repeat
4330    \fi
4331 \endgroup
4332 \bbl@trace{Macros for reading patterns files}
4333 \def\bbl@get@enc#1:#2:#3\@@@{\def\bbl@hyph@enc{#2}}
4334 \ifx\babelcatcodetablenum\@undefined
4335   \ifx\newcatcodetable\@undefined
4336     \def\babelcatcodetablenum{5211}
4337     \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4338   \else
4339     \newcatcodetable\babelcatcodetablenum
4340     \newcatcodetable\bbl@pattcodes
4341   \fi
4342 \else
4343   \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4344 \fi
4345 \def\bbl@luapatterns#1#2{%
4346   \bbl@get@enc#1::\@@@
4347   \setbox\z@\hbox\bgroup
4348     \begingroup
4349       \savecatcodetable\babelcatcodetablenum\relax
4350       \initcatcodetable\bbl@pattcodes\relax
4351       \catcodetable\bbl@pattcodes\relax
4352         \catcode`\#=6  \catcode`\$=3 \catcode`\&=4 \catcode`\^=7
4353         \catcode`\_=8  \catcode`\{=1 \catcode`\}=2 \catcode`\~=13
4354         \catcode`\@=11 \catcode`\^^I=10 \catcode`\^^J=12
4355         \catcode`\<=12 \catcode`\>=12 \catcode`\*=12 \catcode`\.=12
4356         \catcode`\-=12 \catcode`\/=12 \catcode`\[=12 \catcode`\]=12
4357         \catcode`\`=12 \catcode`\'=12 \catcode`\"=12
4358         \input #1\relax
4359       \catcodetable\babelcatcodetablenum\relax
4360     \endgroup
4361     \def\bbl@tempa{#2}%
4362     \ifx\bbl@tempa\@empty\else
4363       \input #2\relax
4364     \fi
4365   \egroup}%
4366 \def\bbl@patterns@lua#1{%
4367   \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
4368     \csname l@#1\endcsname
4369     \edef\bbl@tempa{#1}%
4370   \else
4371     \csname l@#1:\f@encoding\endcsname
4372     \edef\bbl@tempa{#1:\f@encoding}%
4373   \fi\relax
4374   \@namedef{lu@texhyphen@loaded@\the\language}{}% Temp
4375   \@ifundefined{bbl@hyphendata@\the\language}%
4376     {\def\bbl@elt##1##2##3##4{%
4377        \ifnum##2=\csname l@\bbl@tempa\endcsname % #2=spanish, dutch:OT1...
4378          \def\bbl@tempb{##3}%
```

```
4379          \ifx\bbl@tempb\@empty\else % if not a synonymous
4380            \def\bbl@tempc{{##3}{##4}}%
4381          \fi
4382          \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
4383        \fi}%
4384      \bbl@languages
4385      \@ifundefined{bbl@hyphendata@\the\language}%
4386        {\bbl@info{No hyphenation patterns were set for\\%
4387                  language '\bbl@tempa'. Reported}}%
4388        {\expandafter\expandafter\expandafter\bbl@luapatterns
4389          \csname bbl@hyphendata@\the\language\endcsname}}{}}
4390 \endinput\fi
4391  % Here ends \ifx\AddBabelHook\@undefined
4392  % A few lines are only read by hyphen.cfg
4393 \ifx\DisableBabelHook\@undefined
4394  \AddBabelHook{luatex}{everylanguage}{%
4395    \def\process@language##1##2##3{%
4396      \def\process@line####1####2 ####3 ####4 {}}}
4397  \AddBabelHook{luatex}{loadpatterns}{%
4398      \input #1\relax
4399      \expandafter\gdef\csname bbl@hyphendata@\the\language\endcsname
4400        {{#1}{}}}
4401  \AddBabelHook{luatex}{loadexceptions}{%
4402      \input #1\relax
4403      \def\bbl@tempb##1##2{{##1}{#1}}%
4404      \expandafter\xdef\csname bbl@hyphendata@\the\language\endcsname
4405        {\expandafter\expandafter\expandafter\bbl@tempb
4406          \csname bbl@hyphendata@\the\language\endcsname}}
4407 \endinput\fi
4408  % Here stops reading code for hyphen.cfg
4409  % The following is read the 2nd time it's loaded
4410 \begingroup
4411 \catcode`\%=12
4412 \catcode`\'=12
4413 \catcode`\"=12
4414 \catcode`\:=12
4415 \directlua{
4416  Babel = Babel or {}
4417  function Babel.bytes(line)
4418    return line:gsub("(.)",
4419      function (chr) return unicode.utf8.char(string.byte(chr)) end)
4420  end
4421  function Babel.begin_process_input()
4422    if luatexbase and luatexbase.add_to_callback then
4423      luatexbase.add_to_callback('process_input_buffer',
4424                                  Babel.bytes,'Babel.bytes')
4425    else
4426      Babel.callback = callback.find('process_input_buffer')
4427      callback.register('process_input_buffer',Babel.bytes)
4428    end
4429  end
4430  function Babel.end_process_input ()
4431    if luatexbase and luatexbase.remove_from_callback then
4432      luatexbase.remove_from_callback('process_input_buffer','Babel.bytes')
4433    else
4434      callback.register('process_input_buffer',Babel.callback)
4435    end
4436  end
4437  function Babel.addpatterns(pp, lg)
```

```
4438    local lg = lang.new(lg)
4439    local pats = lang.patterns(lg) or ''
4440    lang.clear_patterns(lg)
4441    for p in pp:gmatch('[^%s]+') do
4442      ss = ''
4443      for i in string.utfcharacters(p:gsub('%d', '')) do
4444        ss = ss .. '%d?' .. i
4445      end
4446      ss = ss:gsub('^%%d%?%.', '%%.') .. '%d?'
4447      ss = ss:gsub('%.%%d%?$', '%%.')
4448      pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
4449      if n == 0 then
4450        tex.sprint(
4451          [[\string\csname\space bbl@info\endcsname{New pattern: ]]
4452          .. p .. [[}]])
4453        pats = pats .. ' ' .. p
4454      else
4455        tex.sprint(
4456          [[\string\csname\space bbl@info\endcsname{Renew pattern: ]]
4457          .. p .. [[}]])
4458      end
4459    end
4460    lang.patterns(lg, pats)
4461  end
4462 }
4463 \endgroup
4464 \ifx\newattribute\@undefined\else
4465   \newattribute\bbl@attr@locale
4466   \AddBabelHook{luatex}{beforeextras}{%
4467     \setattribute\bbl@attr@locale\localeid}
4468 \fi
4469 \def\BabelStringsDefault{unicode}
4470 \let\luabbl@stop\relax
4471 \AddBabelHook{luatex}{encodedcommands}{%
4472   \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
4473   \ifx\bbl@tempa\bbl@tempb\else
4474     \directlua{Babel.begin_process_input()}%
4475     \def\luabbl@stop{%
4476       \directlua{Babel.end_process_input()}}%
4477   \fi}%
4478 \AddBabelHook{luatex}{stopcommands}{%
4479   \luabbl@stop
4480   \let\luabbl@stop\relax}
4481 \AddBabelHook{luatex}{patterns}{%
4482   \@ifundefined{bbl@hyphendata@\the\language}%
4483     {\def\bbl@elt##1##2##3##4{%
4484       \ifnum##2=\csname l@##2\endcsname % #2=spanish, dutch:OT1...
4485        \def\bbl@tempb{##3}%
4486        \ifx\bbl@tempb\@empty\else % if not a synonymous
4487          \def\bbl@tempc{{##3}{##4}}%
4488        \fi
4489        \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
4490      \fi}%
4491     \bbl@languages
4492     \@ifundefined{bbl@hyphendata@\the\language}%
4493       {\bbl@info{No hyphenation patterns were set for\\%
4494                 language '#2'. Reported}}%
4495       {\expandafter\expandafter\expandafter\bbl@luapatterns
4496          \csname bbl@hyphendata@\the\language\endcsname}}{}%
```

```
4497    \@ifundefined{bbl@patterns@}{}{%
4498      \begingroup
4499        \bbl@xin@{,\number\language,}{,\bbl@pttnlist}%
4500        \ifin@\else
4501          \ifx\bbl@patterns@\@empty\else
4502            \directlua{ Babel.addpatterns(
4503              [[\bbl@patterns@]], \number\language) }%
4504          \fi
4505          \@ifundefined{bbl@patterns@#1}%
4506            \@empty
4507            {\directlua{ Babel.addpatterns(
4508                [[\space\csname bbl@patterns@#1\endcsname]],
4509                \number\language) }}%
4510          \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
4511        \fi
4512      \endgroup}%
4513    \bbl@exp{%
4514      \bbl@ifunset{bbl@prehc@\languagename}{}%
4515        {\\\bbl@ifblank{\bbl@cs{prehc@\languagename}}{}%
4516          {\prehyphenchar=\bbl@cl{prehc}\relax}}}}
```

\babelpatterns This macro adds patterns. Two macros are used to store them: \bbl@patterns@ for the
global ones and \bbl@patterns@<lang> for language ones. We make sure there is a space
between words when multiple commands are used.

```
4517  \@onlypreamble\babelpatterns
4518  \AtEndOfPackage{%
4519    \newcommand\babelpatterns[2][\@empty]{%
4520      \ifx\bbl@patterns@\relax
4521        \let\bbl@patterns@\@empty
4522      \fi
4523      \ifx\bbl@pttnlist\@empty\else
4524        \bbl@warning{%
4525          You must not intermingle \string\selectlanguage\space and\\%
4526          \string\babelpatterns\space or some patterns will not\\%
4527          be taken into account. Reported}%
4528      \fi
4529      \ifx\@empty#1%
4530        \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
4531      \else
4532        \edef\bbl@tempb{\zap@space#1 \@empty}%
4533        \bbl@for\bbl@tempa\bbl@tempb{%
4534          \bbl@fixname\bbl@tempa
4535          \bbl@iflanguage\bbl@tempa{%
4536            \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
4537              \@ifundefined{bbl@patterns@\bbl@tempa}%
4538                \@empty
4539                {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
4540              #2}}}%
4541      \fi}}
```

13.4   Southeast Asian scripts

First, some general code for line breaking, used by \babelposthyphenation.
In progress. Replace regular (ie, implicit) discretionaries by spaceskips, based on the
previous glyph (which I think makes sense, because the hyphen and the previous char go
always together). Other discretionaries are not touched.
For the moment, only 3 SA languages are activated by default (see Unicode UAX 14).

```
4542 \directlua{
4543   Babel = Babel or {}
4544   Babel.linebreaking = Babel.linebreaking or {}
4545   Babel.linebreaking.before = {}
4546   Babel.linebreaking.after = {}
4547   Babel.locale = {} % Free to use, indexed with \localeid
4548   function Babel.linebreaking.add_before(func)
4549     tex.print([[\noexpand\csname bbl@luahyphenate\endcsname]])
4550     table.insert(Babel.linebreaking.before , func)
4551   end
4552   function Babel.linebreaking.add_after(func)
4553     tex.print([[\noexpand\csname bbl@luahyphenate\endcsname]])
4554     table.insert(Babel.linebreaking.after, func)
4555   end
4556 }
4557 \def\bbl@intraspace#1 #2 #3\@@{%
4558   \directlua{
4559     Babel = Babel or {}
4560     Babel.intraspaces = Babel.intraspaces or {}
4561     Babel.intraspaces['\csname bbl@sbcp@\languagename\endcsname'] = %
4562         {b = #1, p = #2, m = #3}
4563     Babel.locale_props[\the\localeid].intraspace = %
4564         {b = #1, p = #2, m = #3}
4565   }}
4566 \def\bbl@intrapenalty#1\@@{%
4567   \directlua{
4568     Babel = Babel or {}
4569     Babel.intrapenalties = Babel.intrapenalties or {}
4570     Babel.intrapenalties['\csname bbl@sbcp@\languagename\endcsname'] = #1
4571     Babel.locale_props[\the\localeid].intrapenalty = #1
4572   }}
4573 \begingroup
4574 \catcode`\%=12
4575 \catcode`\^=14
4576 \catcode`\'=12
4577 \catcode`\~=12
4578 \gdef\bbl@seaintraspace{^
4579   \let\bbl@seaintraspace\relax
4580   \directlua{
4581     Babel = Babel or {}
4582     Babel.sea_enabled = true
4583     Babel.sea_ranges = Babel.sea_ranges or {}
4584     function Babel.set_chranges (script, chrng)
4585       local c = 0
4586       for s, e in string.gmatch(chrng..' ', '(.-)%.%.(.-)%s') do
4587         Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
4588         c = c + 1
4589       end
4590     end
4591     function Babel.sea_disc_to_space (head)
4592       local sea_ranges = Babel.sea_ranges
4593       local last_char = nil
4594       local quad = 655360      ^^ 10 pt = 655360 = 10 * 65536
4595       for item in node.traverse(head) do
4596         local i = item.id
4597         if i == node.id'glyph' then
4598           last_char = item
4599         elseif i == 7 and item.subtype == 3 and last_char
4600             and last_char.char > 0x0C99 then
```

164

```
4601            quad = font.getfont(last_char.font).size
4602            for lg, rg in pairs(sea_ranges) do
4603              if last_char.char > rg[1] and last_char.char < rg[2] then
4604                lg = lg:sub(1, 4)  ^^ Remove trailing number of, eg, Cyrl1
4605                local intraspace = Babel.intraspaces[lg]
4606                local intrapenalty = Babel.intrapenalties[lg]
4607                local n
4608                if intrapenalty ~= 0 then
4609                  n = node.new(14, 0)     ^^ penalty
4610                  n.penalty = intrapenalty
4611                  node.insert_before(head, item, n)
4612                end
4613                n = node.new(12, 13)     ^^ (glue, spaceskip)
4614                node.setglue(n, intraspace.b * quad,
4615                              intraspace.p * quad,
4616                              intraspace.m * quad)
4617                node.insert_before(head, item, n)
4618                node.remove(head, item)
4619              end
4620            end
4621          end
4622        end
4623      end
4624    }^^
4625    \bbl@luahyphenate}
4626 \catcode`\%=14
4627 \gdef\bbl@cjkintraspace{%
4628   \let\bbl@cjkintraspace\relax
4629   \directlua{
4630     Babel = Babel or {}
4631     require'babel-data-cjk.lua'
4632     Babel.cjk_enabled = true
4633     function Babel.cjk_linebreak(head)
4634       local GLYPH = node.id'glyph'
4635       local last_char = nil
4636       local quad = 655360     % 10 pt = 655360 = 10 * 65536
4637       local last_class = nil
4638       local last_lang = nil
4639
4640       for item in node.traverse(head) do
4641         if item.id == GLYPH then
4642
4643           local lang = item.lang
4644
4645           local LOCALE = node.get_attribute(item,
4646                 luatexbase.registernumber'bbl@attr@locale')
4647           local props = Babel.locale_props[LOCALE]
4648
4649           local class = Babel.cjk_class[item.char].c
4650
4651           if class == 'cp' then class = 'cl' end % )] as CL
4652           if class == 'id' then class = 'I' end
4653
4654           local br = 0
4655           if class and last_class and Babel.cjk_breaks[last_class][class] then
4656             br = Babel.cjk_breaks[last_class][class]
4657           end
4658
4659           if br == 1 and props.linebreak == 'c' and
```

```
4660                lang ~= \the\l@nohyphenation\space and
4661                last_lang ~= \the\l@nohyphenation then
4662              local intrapenalty = props.intrapenalty
4663              if intrapenalty ~= 0 then
4664                local n = node.new(14, 0)      % penalty
4665                n.penalty = intrapenalty
4666                node.insert_before(head, item, n)
4667              end
4668              local intraspace = props.intraspace
4669              local n = node.new(12, 13)      % (glue, spaceskip)
4670              node.setglue(n, intraspace.b * quad,
4671                              intraspace.p * quad,
4672                              intraspace.m * quad)
4673              node.insert_before(head, item, n)
4674            end
4675
4676            quad = font.getfont(item.font).size
4677            last_class = class
4678            last_lang = lang
4679          else % if penalty, glue or anything else
4680            last_class = nil
4681          end
4682        end
4683        lang.hyphenate(head)
4684      end
4685  }%
4686  \bbl@luahyphenate}
4687 \gdef\bbl@luahyphenate{%
4688  \let\bbl@luahyphenate\relax
4689  \directlua{
4690    luatexbase.add_to_callback('hyphenate',
4691    function (head, tail)
4692      if Babel.linebreaking.before then
4693        for k, func in ipairs(Babel.linebreaking.before)  do
4694          func(head)
4695        end
4696      end
4697      if Babel.cjk_enabled then
4698        Babel.cjk_linebreak(head)
4699      end
4700      lang.hyphenate(head)
4701      if Babel.linebreaking.after then
4702        for k, func in ipairs(Babel.linebreaking.after)  do
4703          func(head)
4704        end
4705      end
4706      if Babel.sea_enabled then
4707        Babel.sea_disc_to_space(head)
4708      end
4709    end,
4710    'Babel.hyphenate')
4711  }
4712 }
4713 \endgroup
4714 \def\bbl@provide@intraspace{%
4715  \bbl@ifunset{bbl@intsp@\languagename}{}%
4716    {\expandafter\ifx\csname bbl@intsp@\languagename\endcsname\@empty\else
4717      \bbl@xin@{\bbl@cl{lnbrk}}{c}%
4718      \ifin@           % cjk
```

```
4719        \bbl@cjkintraspace
4720        \directlua{
4721            Babel = Babel or {}
4722            Babel.locale_props = Babel.locale_props or {}
4723            Babel.locale_props[\the\localeid].linebreak = 'c'
4724        }%
4725        \bbl@exp{\\\bbl@intraspace\bbl@cl{intsp}\\\@@}%
4726        \ifx\bbl@KVP@intrapenalty\@nil
4727          \bbl@intrapenalty0\@@
4728        \fi
4729      \else            % sea
4730        \bbl@seaintraspace
4731        \bbl@exp{\\\bbl@intraspace\bbl@cl{intsp}\\\@@}%
4732        \directlua{
4733            Babel = Babel or {}
4734            Babel.sea_ranges = Babel.sea_ranges or {}
4735            Babel.set_chranges('\bbl@cl{sbcp}',
4736                               '\bbl@cl{chrng}')
4737        }%
4738        \ifx\bbl@KVP@intrapenalty\@nil
4739          \bbl@intrapenalty0\@@
4740        \fi
4741      \fi
4742    \fi
4743    \ifx\bbl@KVP@intrapenalty\@nil\else
4744      \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
4745    \fi}}
```

13.5 CJK line breaking

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

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

Work in progress.

Common stuff.

```
4746 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4747 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
4748 \DisableBabelHook{babel-fontspec}
4749 ⟨⟨Font selection⟩⟩
```

13.6 Automatic fonts and ids switching

After defining the blocks for a number of scripts (must be extended and very likely fine tuned), we define a short function which just traverse the node list to carry out the replacements. The table loc_to_scr gets the locale form a script range (note the locale is the key, and that there is an intermediate table built on the fly for optimization). This locale is then used to get the \language and the \localeid as stored in locale_props, as well as the font (as requested). In the latter table a key starting with / maps the font from the global one (the key) to the local one (the value). Maths are skipped and discretionaries are handled in a special way.

```
4750 \directlua{
4751 Babel.script_blocks = {
```

```
4752    ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
4753                {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
4754    ['Armn'] = {{0x0530, 0x058F}},
4755    ['Beng'] = {{0x0980, 0x09FF}},
4756    ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
4757    ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
4758    ['Cyrl'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
4759                {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
4760    ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
4761    ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
4762                {0xAB00, 0xAB2F}},
4763    ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
4764    % Don't follow strictly Unicode, which places some Coptic letters in
4765    % the 'Greek and Coptic' block
4766    ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
4767    ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
4768                {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
4769                {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
4770                {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
4771                {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
4772                {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
4773    ['Hebr'] = {{0x0590, 0x05FF}},
4774    ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
4775                {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
4776    ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
4777    ['Knda'] = {{0x0C80, 0x0CFF}},
4778    ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
4779                {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
4780                {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
4781    ['Laoo'] = {{0x0E80, 0x0EFF}},
4782    ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
4783                {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
4784                {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
4785    ['Mahj'] = {{0x11150, 0x1117F}},
4786    ['Mlym'] = {{0x0D00, 0x0D7F}},
4787    ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
4788    ['Orya'] = {{0x0B00, 0x0B7F}},
4789    ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
4790    ['Syrc'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
4791    ['Taml'] = {{0x0B80, 0x0BFF}},
4792    ['Telu'] = {{0x0C00, 0x0C7F}},
4793    ['Tfng'] = {{0x2D30, 0x2D7F}},
4794    ['Thai'] = {{0x0E00, 0x0E7F}},
4795    ['Tibt'] = {{0x0F00, 0x0FFF}},
4796    ['Vaii'] = {{0xA500, 0xA63F}},
4797    ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
4798 }
4799
4800 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
4801 Babel.script_blocks.Hant = Babel.script_blocks.Hans
4802 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
4803
4804 function Babel.locale_map(head)
4805   if not Babel.locale_mapped then return head end
4806
4807   local LOCALE = luatexbase.registernumber'bbl@attr@locale'
4808   local GLYPH = node.id('glyph')
4809   local inmath = false
4810   local toloc_save
```

```
4811    for item in node.traverse(head) do
4812      local toloc
4813      if not inmath and item.id == GLYPH then
4814        % Optimization: build a table with the chars found
4815        if Babel.chr_to_loc[item.char] then
4816          toloc = Babel.chr_to_loc[item.char]
4817        else
4818          for lc, maps in pairs(Babel.loc_to_scr) do
4819            for _, rg in pairs(maps) do
4820              if item.char >= rg[1] and item.char <= rg[2] then
4821                Babel.chr_to_loc[item.char] = lc
4822                toloc = lc
4823                break
4824              end
4825            end
4826          end
4827        end
4828        % Now, take action, but treat composite chars in a different
4829        % fashion, because they 'inherit' the previous locale. Not yet
4830        % optimized.
4831        if not toloc and
4832          (item.char >= 0x0300 and item.char <= 0x036F) or
4833          (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
4834          (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
4835          toloc = toloc_save
4836        end
4837        if toloc and toloc > -1 then
4838          if Babel.locale_props[toloc].lg then
4839            item.lang = Babel.locale_props[toloc].lg
4840            node.set_attribute(item, LOCALE, toloc)
4841          end
4842          if Babel.locale_props[toloc]['/'..item.font] then
4843            item.font = Babel.locale_props[toloc]['/'..item.font]
4844          end
4845          toloc_save = toloc
4846        end
4847      elseif not inmath and item.id == 7 then
4848        item.replace = item.replace and Babel.locale_map(item.replace)
4849        item.pre     = item.pre and Babel.locale_map(item.pre)
4850        item.post    = item.post and Babel.locale_map(item.post)
4851      elseif item.id == node.id'math' then
4852        inmath = (item.subtype == 0)
4853      end
4854    end
4855    return head
4856 end
4857 }
```

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

```
4858 \newcommand\babelcharproperty[1]{%
4859   \count@=#1\relax
4860   \ifvmode
4861     \expandafter\bbl@chprop
4862   \else
4863     \bbl@error{\string\babelcharproperty\space can be used only in\\%
4864               vertical mode (preamble or between paragraphs)}%
4865               {See the manual for futher info}%
4866   \fi}
```

```
4867 \newcommand\bbl@chprop[3][\the\count@]{%
4868   \@tempcnta=#1\relax
4869   \bbl@ifunset{bbl@chprop@#2}%
4870     {\bbl@error{No property named '#2'. Allowed values are\\%
4871                 direction (bc), mirror (bmg), and linebreak (lb)}%
4872                 {See the manual for futher info}}%
4873     {}%
4874   \loop
4875     \bbl@cs{chprop@#2}{#3}%
4876   \ifnum\count@<\@tempcnta
4877     \advance\count@\@ne
4878   \repeat}
4879 \def\bbl@chprop@direction#1{%
4880   \directlua{
4881     Babel.characters[\the\count@] =  Babel.characters[\the\count@] or {}
4882     Babel.characters[\the\count@]['d'] = '#1'
4883   }}
4884 \let\bbl@chprop@bc\bbl@chprop@direction
4885 \def\bbl@chprop@mirror#1{%
4886   \directlua{
4887     Babel.characters[\the\count@] =  Babel.characters[\the\count@] or {}
4888     Babel.characters[\the\count@]['m'] = '\number#1'
4889   }}
4890 \let\bbl@chprop@bmg\bbl@chprop@mirror
4891 \def\bbl@chprop@linebreak#1{%
4892   \directlua{
4893     Babel.Babel.cjk_characters[\the\count@] = Babel.Babel.cjk_characters[\the\count@] or {}
4894     Babel.Babel.cjk_characters[\the\count@]['c'] = '#1'
4895   }}
4896 \let\bbl@chprop@lb\bbl@chprop@linebreak
4897 \def\bbl@chprop@locale#1{%
4898   \directlua{
4899     Babel.chr_to_loc = Babel.chr_to_loc or {}
4900     Babel.chr_to_loc[\the\count@] =
4901       \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@@#1}}\space
4902   }}
```

Post-handling hyphenation patterns for non-standard rules, like ff to ff-f. There are still some issues with speed (not very slow, but still slow).

After declaring the table containing the patterns with their replacements, we define some auxiliary functions: str_to_nodes converts the string returned by a function to a node list, taking the node at base as a model (font, language, etc.); fetch_word fetches a series of glyphs and discretionaries, which pattern is matched against (if there is a match, it is called again before trying other patterns, and this is very likely the main bottleneck). post_hyphenate_replace is the callback applied after lang.hyphenate. This means the automatic hyphenation points are known. As empty captures return a byte position (as explained in the luatex manual), we must convert it to a utf8 position. With first, the last byte can be the leading byte in a utf8 sequence, so we just remove it and add 1 to the resulting length. With last we must take into account the capture position points to the next character. Here word_head points to the starting node of the text to be matched.

```
4903 \begingroup
4904 \catcode`\#=12
4905 \catcode`\%=12
4906 \catcode`\&=14
4907 \directlua{
4908   Babel.linebreaking.replacements = {}
4909
4910   function Babel.str_to_nodes(fn, matches, base)
```

```
4911    local n, head, last
4912    if fn == nil then return nil end
4913    for s in string.utfvalues(fn(matches)) do
4914      if base.id == 7 then
4915        base = base.replace
4916      end
4917      n = node.copy(base)
4918      n.char    = s
4919      if not head then
4920        head = n
4921      else
4922        last.next = n
4923      end
4924      last = n
4925    end
4926    return head
4927  end
4928
4929  function Babel.fetch_word(head, funct)
4930    local word_string = ''
4931    local word_nodes = {}
4932    local lang
4933    local item = head
4934
4935    while item do
4936
4937      if item.id == 29
4938          and not(item.char == 124) &% ie, not |
4939          and not(item.char == 61)  &% ie, not =
4940          and (item.lang == lang or lang == nil) then
4941        lang = lang or item.lang
4942        word_string = word_string .. unicode.utf8.char(item.char)
4943        word_nodes[#word_nodes+1] = item
4944
4945      elseif item.id == 7 and item.subtype == 2 then
4946        word_string = word_string .. '='
4947        word_nodes[#word_nodes+1] = item
4948
4949      elseif item.id == 7 and item.subtype == 3 then
4950        word_string = word_string .. '|'
4951        word_nodes[#word_nodes+1] = item
4952
4953      elseif word_string == '' then
4954        &% pass
4955
4956      else
4957        return word_string, word_nodes, item, lang
4958      end
4959
4960      item = item.next
4961    end
4962  end
4963
4964  function Babel.post_hyphenate_replace(head)
4965    local u = unicode.utf8
4966    local lbkr = Babel.linebreaking.replacements
4967    local word_head = head
4968
4969    while true do
```

```
4970      local w, wn, nw, lang = Babel.fetch_word(word_head)
4971      if not lang then return head end
4972
4973      if not lbkr[lang] then
4974        break
4975      end
4976
4977      for k=1, #lbkr[lang] do
4978        local p = lbkr[lang][k].pattern
4979        local r = lbkr[lang][k].replace
4980
4981        while true do
4982          local matches = { u.match(w, p) }
4983          if #matches < 2 then break end
4984
4985          local first = table.remove(matches, 1)
4986          local last =  table.remove(matches, #matches)
4987
4988          &% Fix offsets, from bytes to unicode.
4989          first = u.len(w:sub(1, first-1)) + 1
4990          last  = u.len(w:sub(1, last-1))
4991
4992          local new  &% used when inserting and removing nodes
4993          local changed = 0
4994
4995          &% This loop traverses the replace list and takes the
4996          &% corresponding actions
4997          for q = first, last do
4998            local crep = r[q-first+1]
4999            local char_node = wn[q]
5000            local char_base = char_node
5001
5002            if crep and crep.data then
5003              char_base = wn[crep.data+first-1]
5004            end
5005
5006            if crep == {} then
5007              break
5008            elseif crep == nil then
5009              changed = changed + 1
5010              node.remove(head, char_node)
5011            elseif crep and (crep.pre or crep.no or crep.post) then
5012              changed = changed + 1
5013              d = node.new(7, 0)   &% (disc, discretionary)
5014              d.pre = Babel.str_to_nodes(crep.pre, matches, char_base)
5015              d.post = Babel.str_to_nodes(crep.post, matches, char_base)
5016              d.replace = Babel.str_to_nodes(crep.no, matches, char_base)
5017              d.attr = char_base.attr
5018              if crep.pre == nil then  &% TeXbook p96
5019                d.penalty  = crep.penalty or tex.hyphenpenalty
5020              else
5021                d.penalty  = crep.penalty or tex.exhyphenpenalty
5022              end
5023              head, new = node.insert_before(head, char_node, d)
5024              node.remove(head, char_node)
5025              if q == 1 then
5026                word_head = new
5027              end
5028            elseif crep and crep.string then
```

```
5029                changed = changed + 1
5030                local str = crep.string(matches)
5031                if str == '' then
5032                  if q == 1 then
5033                    word_head = char_node.next
5034                  end
5035                  head, new = node.remove(head, char_node)
5036                elseif char_node.id == 29 and u.len(str) == 1 then
5037                  char_node.char = string.utfvalue(str)
5038                else
5039                  local n
5040                  for s in string.utfvalues(str) do
5041                    if char_node.id == 7 then
5042                      log('Automatic hyphens cannot be replaced, just removed.')
5043                    else
5044                      n = node.copy(char_base)
5045                    end
5046                    n.char = s
5047                    if q == 1 then
5048                      head, new = node.insert_before(head, char_node, n)
5049                      word_head = new
5050                    else
5051                      node.insert_before(head, char_node, n)
5052                    end
5053                  end
5054
5055                  node.remove(head, char_node)
5056                end   &% string length
5057              end   &% if char and char.string
5058            end   &% for char in match
5059            if changed > 20 then
5060              texio.write('Too many changes. Ignoring the rest.')
5061            elseif changed > 0 then
5062              w, wn, nw = Babel.fetch_word(word_head)
5063            end
5064
5065        end   &% for match
5066      end   &% for patterns
5067      word_head = nw
5068    end   &% for words
5069    return head
5070  end
5071
5072  &% The following functions belong to the next macro
5073
5074  &% This table stores capture maps, numbered consecutively
5075  Babel.capture_maps = {}
5076
5077  function Babel.capture_func(key, cap)
5078    local ret = "[[" .. cap:gsub('{([0-9])}', "]]..m[%1]..[[") .. "]]"
5079    ret = ret:gsub('{([0-9])|([^|]+)|(.-)}', Babel.capture_func_map)
5080    ret = ret:gsub("%[%[%]%.%.", '')
5081    ret = ret:gsub("%.%.%[%[%]%]", '')
5082    return key .. [[=function(m) return ]] .. ret .. [[ end]]
5083  end
5084
5085  function Babel.capt_map(from, mapno)
5086    return Babel.capture_maps[mapno][from] or from
5087  end
```

```
5088
5089    &% Handle the {n|abc|ABC} syntax in captures
5090    function Babel.capture_func_map(capno, from, to)
5091      local froms = {}
5092      for s in string.utfcharacters(from) do
5093        table.insert(froms, s)
5094      end
5095      local cnt = 1
5096      table.insert(Babel.capture_maps, {})
5097      local mlen = table.getn(Babel.capture_maps)
5098      for s in string.utfcharacters(to) do
5099        Babel.capture_maps[mlen][froms[cnt]] = s
5100        cnt = cnt + 1
5101      end
5102      return "]]..Babel.capt_map(m[" .. capno .. "]," ..
5103             (mlen) .. ").." .. "[["
5104    end
5105
5106  }
```

Now the T_EX high level interface, which requires the function defined above for converting strings to functions returning a string. These functions handle the {*n*} syntax. For example, `pre={1}{1}-` becomes `function(m) return m[1]..m[1]..'-' end`, where `m` are the matches returned after applying the pattern. With a mapped capture the functions are similar to `function(m) return Babel.capt_map(m[1],1) end`, where the last argument identifies the mapping to be applied to `m[1]`. The way it is carried out is somewhat tricky, but the effect in not dissimilar to lua `load` – save the code as string in a TeX macro, and expand this macro at the appropriate place. As `\directlua` does not take into account the current catcode of `@`, we just avoid this character in macro names (which explains the internal group, too).

```
5107 \catcode`\#=6
5108 \gdef\babelposthyphenation#1#2#3{&%
5109   \bbl@activateposthyphen
5110   \begingroup
5111     \def\babeltempa{\bbl@add@list\babeltempb}&%
5112     \let\babeltempb\@empty
5113     \bbl@foreach{#3}{&%
5114       \bbl@ifsamestring{##1}{remove}&%
5115         {\bbl@add@list\babeltempb{nil}}&%
5116         {\directlua{
5117            local rep = [[##1]]
5118            rep = rep:gsub(    '(no)%s*=%s*([^%s,]*)', Babel.capture_func)
5119            rep = rep:gsub(   '(pre)%s*=%s*([^%s,]*)', Babel.capture_func)
5120            rep = rep:gsub(  '(post)%s*=%s*([^%s,]*)', Babel.capture_func)
5121            rep = rep:gsub('(string)%s*=%s*([^%s,]*)', Babel.capture_func)
5122            tex.print([[\string\babeltempa{{]] .. rep .. [[}}]])
5123          }}}&%
5124     \directlua{
5125       local lbkr = Babel.linebreaking.replacements
5126       local u = unicode.utf8
5127       &% Convert pattern:
5128       local patt = string.gsub([[#2]], '%s', '')
5129       if not u.find(patt, '()', nil, true) then
5130         patt = '()' .. patt .. '()'
5131       end
5132       patt = u.gsub(patt, '{(.)}',
5133                function (n)
5134                  return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
```

```
5135              end)
5136         lbkr[\the\csname l@#1\endcsname] = lbkr[\the\csname l@#1\endcsname] or {}
5137         table.insert(lbkr[\the\csname l@#1\endcsname],
5138                      { pattern = patt, replace = { \babeltempb } })
5139     }&%
5140   \endgroup}
5141 \endgroup
5142 \def\bbl@activateposthyphen{%
5143   \let\bbl@activateposthyphen\relax
5144   \directlua{
5145     Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
5146   }}
```

13.7 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.) with `bidi=basic`, without having to patch almost any macro where text direction is relevant.

`\@hangfrom` is useful in many contexts and it is redefined always with the `layout` option. There are, however, a number of issues when the text direction is not the same as the box direction (as set by `\bodydir`), and when `\parbox` and `\hangindent` are involved. Fortunately, latest releases of luatex simplify a lot the solution with `\shapemode`.

With the issue #15 I realized commands are best patched, instead of redefined. With a few lines, a modification could be applied to several classes and packages. Now, `tabular` seems to work (at least in simple cases) with array, tabularx, hhline, colortbl, longtable, booktabs, etc. However, dcolumn still fails.

```
5147 \bbl@trace{Redefinitions for bidi layout}
5148 \ifx\@eqnnum\@undefined\else
5149   \ifx\bbl@attr@dir\@undefined\else
5150     \edef\@eqnnum{{%
5151       \unexpanded{\ifcase\bbl@attr@dir\else\bbl@textdir\@ne\fi}%
5152       \unexpanded\expandafter{\@eqnnum}}}
5153   \fi
5154 \fi
5155 \ifx\bbl@opt@layout\@nnil\endinput\fi  % if no layout
5156 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
5157   \def\bbl@nextfake#1{%  non-local changes, use always inside a group!
5158     \bbl@exp{%
5159       \mathdir\the\bodydir
5160       #1%                Once entered in math, set boxes to restore values
5161       \<ifmmode>%
5162         \everyvbox{%
5163           \the\everyvbox
5164           \bodydir\the\bodydir
5165           \mathdir\the\mathdir
5166           \everyhbox{\the\everyhbox}%
5167           \everyvbox{\the\everyvbox}}%
5168         \everyhbox{%
5169           \the\everyhbox
5170           \bodydir\the\bodydir
5171           \mathdir\the\mathdir
5172           \everyhbox{\the\everyhbox}%
5173           \everyvbox{\the\everyvbox}}%
5174       \<fi>}}%
5175   \def\@hangfrom#1{%
```

```
5176        \setbox\@tempboxa\hbox{{#1}}%
5177        \hangindent\wd\@tempboxa
5178        \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
5179          \shapemode\@ne
5180        \fi
5181        \noindent\box\@tempboxa}
5182 \fi
5183 \IfBabelLayout{tabular}
5184   {\let\bbl@OL@@tabular\@tabular
5185    \bbl@replace\@tabular{$}{\bbl@nextfake$}%
5186    \let\bbl@NL@@tabular\@tabular
5187    \AtBeginDocument{%
5188      \ifx\bbl@NL@@tabular\@tabular\else
5189        \bbl@replace\@tabular{$}{\bbl@nextfake$}%
5190        \let\bbl@NL@@tabular\@tabular
5191      \fi}}
5192   {}
5193 \IfBabelLayout{lists}
5194   {\let\bbl@OL@list\list
5195    \bbl@sreplace\list{\parshape}{\bbl@listparshape}%
5196    \let\bbl@NL@list\list
5197    \def\bbl@listparshape#1#2#3{%
5198      \parshape #1 #2 #3 %
5199      \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
5200        \shapemode\tw@
5201      \fi}}
5202   {}
5203 \IfBabelLayout{graphics}
5204   {\let\bbl@pictresetdir\relax
5205    \def\bbl@pictsetdir{%
5206      \ifcase\bbl@thetextdir
5207        \let\bbl@pictresetdir\relax
5208      \else
5209        \textdir TLT\relax
5210        \def\bbl@pictresetdir{\textdir TRT\relax}%
5211      \fi}%
5212    \let\bbl@OL@@picture\@picture
5213    \let\bbl@OL@put\put
5214    \bbl@sreplace\@picture{\hskip-}{\bbl@pictsetdir\hskip-}%
5215    \def\put(#1,#2)#3{%  Not easy to patch. Better redefine.
5216      \@killglue
5217      \raise#2\unitlength
5218      \hb@xt@\z@{\kern#1\unitlength{\bbl@pictresetdir#3}\hss}}%
5219    \AtBeginDocument
5220      {\ifx\tikz@atbegin@node\@undefined\else
5221        \let\bbl@OL@pgfpicture\pgfpicture
5222        \bbl@sreplace\pgfpicture{\pgfpicturetrue}{\bbl@pictsetdir\pgfpicturetrue}%
5223        \bbl@add\pgfsys@beginpicture{\bbl@pictsetdir}%
5224        \bbl@add\tikz@atbegin@node{\bbl@pictresetdir}%
5225      \fi}}
5226   {}
```

Implicitly reverses sectioning labels in bidi=basic-r, because the full stop is not in contact with L numbers any more. I think there must be a better way. Assumes bidi=basic, but there are some additional readjustments for bidi=default.

```
5227 \IfBabelLayout{counters}%
5228   {\let\bbl@OL@@textsuperscript\@textsuperscript
5229    \bbl@sreplace\@textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
5230    \let\bbl@latinarabic=\@arabic
```

```
5231    \let\bbl@OL@@arabic\@arabic
5232    \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
5233    \@ifpackagewith{babel}{bidi=default}%
5234      {\let\bbl@asciiroman=\@roman
5235       \let\bbl@OL@@roman\@roman
5236       \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciiroman#1}}}%
5237       \let\bbl@asciiRoman=\@Roman
5238       \let\bbl@OL@@roman\@Roman
5239       \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}%
5240       \let\bbl@OL@labelenumii\labelenumii
5241       \def\labelenumii{)\theenumii(}%
5242       \let\bbl@OL@p@enumiii\p@enumiii
5243       \def\p@enumiii{\p@enumii)\theenumii(}}{}}{}
5244 ⟨⟨Footnote changes⟩⟩
5245 \IfBabelLayout{footnotes}%
5246   {\let\bbl@OL@footnote\footnote
5247    \BabelFootnote\footnote\languagename{}{}%
5248    \BabelFootnote\localfootnote\languagename{}{}%
5249    \BabelFootnote\mainfootnote{}{}{}}
5250   {}
```

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.

```
5251 \IfBabelLayout{extras}%
5252   {\let\bbl@OL@underline\underline
5253    \bbl@sreplace\underline{$\@@underline}{\bbl@nextfake$\@@underline}%
5254    \let\bbl@OL@LaTeX2e\LaTeX2e
5255    \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
5256      \if b\expandafter\@car\f@series\@nil\boldmath\fi
5257      \babelsublr{%
5258        \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}}}
5259   {}
5260 ⟨/luatex⟩
```

13.8 Auto bidi with `basic` and `basic-r`

The file babel-data-bidi.lua currently only contains data. It is a large and boring file and it is not shown here (see the generated file), but here is a sample:

```
[0x25]={d='et'},
[0x26]={d='on'},
[0x27]={d='on'},
[0x28]={d='on', m=0x29},
[0x29]={d='on', m=0x28},
[0x2A]={d='on'},
[0x2B]={d='es'},
[0x2C]={d='cs'},
```

For the meaning of these codes, see the Unicode standard.
Now the `basic-r` bidi mode. One of the aims is to implement a fast and simple bidi algorithm, with a single loop. I managed to do it for R texts, with a second smaller loop for a special case. The code is still somewhat chaotic, but its behavior is essentially correct. I cannot resist copying the following text from Emacs `bidi.c` (which also attempts to implement the bidi algorithm with a single loop):

Arrrgh!! The UAX#9 algorithm is too deeply entrenched in the assumption of batch-style processing [...]. May the fleas of a thousand camels infest the armpits of

177

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.

```
5261 ⟨∗basic-r⟩
5262 Babel = Babel or {}
5263
5264 Babel.bidi_enabled = true
5265
5266 require('babel-data-bidi.lua')
5267
5268 local characters = Babel.characters
5269 local ranges = Babel.ranges
5270
5271 local DIR = node.id("dir")
5272
5273 local function dir_mark(head, from, to, outer)
5274   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
5275   local d = node.new(DIR)
5276   d.dir = '+' .. dir
5277   node.insert_before(head, from, d)
5278   d = node.new(DIR)
5279   d.dir = '-' .. dir
5280   node.insert_after(head, to, d)
5281 end
5282
5283 function Babel.bidi(head, ispar)
5284   local first_n, last_n        -- first and last char with nums
5285   local last_es                -- an auxiliary 'last' used with nums
5286   local first_d, last_d        -- first and last char in L/R block
5287   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):

```
5288   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
5289   local strong_lr = (strong == 'l') and 'l' or 'r'
5290   local outer = strong
5291
5292   local new_dir = false
5293   local first_dir = false
5294   local inmath = false
```

```
5295
5296    local last_lr
5297
5298    local type_n = ''
5299
5300    for item in node.traverse(head) do
5301
5302      -- three cases: glyph, dir, otherwise
5303      if item.id == node.id'glyph'
5304        or (item.id == 7 and item.subtype == 2) then
5305
5306        local itemchar
5307        if item.id == 7 and item.subtype == 2 then
5308          itemchar = item.replace.char
5309        else
5310          itemchar = item.char
5311        end
5312        local chardata = characters[itemchar]
5313        dir = chardata and chardata.d or nil
5314        if not dir then
5315          for nn, et in ipairs(ranges) do
5316            if itemchar < et[1] then
5317              break
5318            elseif itemchar <= et[2] then
5319              dir = et[3]
5320              break
5321            end
5322          end
5323        end
5324        dir = dir or 'l'
5325        if inmath then dir = ('TRT' == tex.mathdir) and 'r' or 'l' end
```

Next is based on the assumption babel sets the language AND switches the script with its dir. We treat a language block as a separate Unicode sequence. The following piece of code is executed at the first glyph after a 'dir' node. We don't know the current language until then. This is not exactly true, as the math mode may insert explicit dirs in the node list, so, for the moment there is a hack by brute force (just above).

```
5326        if new_dir then
5327          attr_dir = 0
5328          for at in node.traverse(item.attr) do
5329            if at.number == luatexbase.registernumber'bbl@attr@dir' then
5330              attr_dir = at.value % 3
5331            end
5332          end
5333          if attr_dir == 1 then
5334            strong = 'r'
5335          elseif attr_dir == 2 then
5336            strong = 'al'
5337          else
5338            strong = 'l'
5339          end
5340          strong_lr = (strong == 'l') and 'l' or 'r'
5341          outer = strong_lr
5342          new_dir = false
5343        end
5344
5345        if dir == 'nsm' then dir = strong end                -- W1
```

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

```
5346        dir_real = dir                     -- We need dir_real to set strong below
5347        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:

```
5348        if strong == 'al' then
5349          if dir == 'en' then dir = 'an' end                  -- W2
5350          if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
5351          strong_lr = 'r'                                     -- W3
5352        end
```

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

```
5353      elseif item.id == node.id'dir' and not inmath then
5354        new_dir = true
5355        dir = nil
5356      elseif item.id == node.id'math' then
5357        inmath = (item.subtype == 0)
5358      else
5359        dir = nil           -- Not a char
5360      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>.

```
5361      if dir == 'en' or dir == 'an' or dir == 'et' then
5362        if dir ~= 'et' then
5363          type_n = dir
5364        end
5365        first_n = first_n or item
5366        last_n = last_es or item
5367        last_es = nil
5368      elseif dir == 'es' and last_n then -- W3+W6
5369        last_es = item
5370      elseif dir == 'cs' then                -- it's right - do nothing
5371      elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
5372        if strong_lr == 'r' and type_n ~= '' then
5373          dir_mark(head, first_n, last_n, 'r')
5374        elseif strong_lr == 'l' and first_d and type_n == 'an' then
5375          dir_mark(head, first_n, last_n, 'r')
5376          dir_mark(head, first_d, last_d, outer)
5377          first_d, last_d = nil, nil
5378        elseif strong_lr == 'l' and type_n ~= '' then
5379          last_d = last_n
5380        end
5381        type_n = ''
5382        first_n, last_n = nil, nil
5383      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:

```
5384      if dir == 'l' or dir == 'r' then
5385        if dir ~= outer then
```

```
5386        first_d = first_d or item
5387        last_d = item
5388      elseif first_d and dir ~= strong_lr then
5389        dir_mark(head, first_d, last_d, outer)
5390        first_d, last_d = nil, nil
5391      end
5392    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.

```
5393    if dir and not last_lr and dir ~= 'l' and outer == 'r' then
5394      item.char = characters[item.char] and
5395                  characters[item.char].m or item.char
5396    elseif (dir or new_dir) and last_lr ~= item then
5397      local mir = outer .. strong_lr .. (dir or outer)
5398      if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
5399        for ch in node.traverse(node.next(last_lr)) do
5400          if ch == item then break end
5401          if ch.id == node.id'glyph' and characters[ch.char] then
5402            ch.char = characters[ch.char].m or ch.char
5403          end
5404        end
5405      end
5406    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`).

```
5407    if dir == 'l' or dir == 'r' then
5408      last_lr = item
5409      strong = dir_real          -- Don't search back - best save now
5410      strong_lr = (strong == 'l') and 'l' or 'r'
5411    elseif new_dir then
5412      last_lr = nil
5413    end
5414  end
```

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

```
5415  if last_lr and outer == 'r' then
5416    for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
5417      if characters[ch.char] then
5418        ch.char = characters[ch.char].m or ch.char
5419      end
5420    end
5421  end
5422  if first_n then
5423    dir_mark(head, first_n, last_n, outer)
5424  end
5425  if first_d then
5426    dir_mark(head, first_d, last_d, outer)
5427  end
```

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

```
5428  return node.prev(head) or head
5429 end
5430 ⟨/basic-r⟩
```

181

And here the Lua code for bidi=basic:

```
5431 ⟨*basic⟩
5432 Babel = Babel or {}
5433
5434 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
5435
5436 Babel.fontmap = Babel.fontmap or {}
5437 Babel.fontmap[0] = {}        -- l
5438 Babel.fontmap[1] = {}        -- r
5439 Babel.fontmap[2] = {}        -- al/an
5440
5441 Babel.bidi_enabled = true
5442 Babel.mirroring_enabled = true
5443
5444 require('babel-data-bidi.lua')
5445
5446 local characters = Babel.characters
5447 local ranges = Babel.ranges
5448
5449 local DIR = node.id('dir')
5450 local GLYPH = node.id('glyph')
5451
5452 local function insert_implicit(head, state, outer)
5453   local new_state = state
5454   if state.sim and state.eim and state.sim ~= state.eim then
5455     dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
5456     local d = node.new(DIR)
5457     d.dir = '+' .. dir
5458     node.insert_before(head, state.sim, d)
5459     local d = node.new(DIR)
5460     d.dir = '-' .. dir
5461     node.insert_after(head, state.eim, d)
5462   end
5463   new_state.sim, new_state.eim = nil, nil
5464   return head, new_state
5465 end
5466
5467 local function insert_numeric(head, state)
5468   local new
5469   local new_state = state
5470   if state.san and state.ean and state.san ~= state.ean then
5471     local d = node.new(DIR)
5472     d.dir = '+TLT'
5473     _, new = node.insert_before(head, state.san, d)
5474     if state.san == state.sim then state.sim = new end
5475     local d = node.new(DIR)
5476     d.dir = '-TLT'
5477     _, new = node.insert_after(head, state.ean, d)
5478     if state.ean == state.eim then state.eim = new end
5479   end
5480   new_state.san, new_state.ean = nil, nil
5481   return head, new_state
5482 end
5483
5484 -- TODO - \hbox with an explicit dir can lead to wrong results
5485 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
5486 -- was s made to improve the situation, but the problem is the 3-dir
5487 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
```

```lua
5488 -- well.
5489
5490 function Babel.bidi(head, ispar, hdir)
5491   local d    -- d is used mainly for computations in a loop
5492   local prev_d = ''
5493   local new_d = false
5494
5495   local nodes = {}
5496   local outer_first = nil
5497   local inmath = false
5498
5499   local glue_d = nil
5500   local glue_i = nil
5501
5502   local has_en = false
5503   local first_et = nil
5504
5505   local ATDIR = luatexbase.registernumber'bbl@attr@dir'
5506
5507   local save_outer
5508   local temp = node.get_attribute(head, ATDIR)
5509   if temp then
5510     temp = temp % 3
5511     save_outer = (temp == 0 and 'l') or
5512                  (temp == 1 and 'r') or
5513                  (temp == 2 and 'al')
5514   elseif ispar then              -- Or error? Shouldn't happen
5515     save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
5516   else                          -- Or error? Shouldn't happen
5517     save_outer = ('TRT' == hdir) and 'r' or 'l'
5518   end
5519     -- when the callback is called, we are just _after_ the box,
5520     -- and the textdir is that of the surrounding text
5521 -- if not ispar and hdir ~= tex.textdir then
5522 --   save_outer = ('TRT' == hdir) and 'r' or 'l'
5523 -- end
5524   local outer = save_outer
5525   local last = outer
5526   -- 'al' is only taken into account in the first, current loop
5527   if save_outer == 'al' then save_outer = 'r' end
5528
5529   local fontmap = Babel.fontmap
5530
5531   for item in node.traverse(head) do
5532
5533     -- In what follows, #node is the last (previous) node, because the
5534     -- current one is not added until we start processing the neutrals.
5535
5536     -- three cases: glyph, dir, otherwise
5537     if item.id == GLYPH
5538        or (item.id == 7 and item.subtype == 2) then
5539
5540       local d_font = nil
5541       local item_r
5542       if item.id == 7 and item.subtype == 2 then
5543         item_r = item.replace    -- automatic discs have just 1 glyph
5544       else
5545         item_r = item
5546       end
```

```
5547        local chardata = characters[item_r.char]
5548        d = chardata and chardata.d or nil
5549        if not d or d == 'nsm' then
5550          for nn, et in ipairs(ranges) do
5551            if item_r.char < et[1] then
5552              break
5553            elseif item_r.char <= et[2] then
5554              if not d then d = et[3]
5555              elseif d == 'nsm' then d_font = et[3]
5556              end
5557              break
5558            end
5559          end
5560        end
5561        d = d or 'l'
5562
5563        -- A short 'pause' in bidi for mapfont
5564        d_font = d_font or d
5565        d_font = (d_font == 'l' and 0) or
5566                 (d_font == 'nsm' and 0) or
5567                 (d_font == 'r' and 1) or
5568                 (d_font == 'al' and 2) or
5569                 (d_font == 'an' and 2) or nil
5570        if d_font and fontmap and fontmap[d_font][item_r.font] then
5571          item_r.font = fontmap[d_font][item_r.font]
5572        end
5573
5574        if new_d then
5575          table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
5576          if inmath then
5577            attr_d = 0
5578          else
5579            attr_d = node.get_attribute(item, ATDIR)
5580            attr_d = attr_d % 3
5581          end
5582          if attr_d == 1 then
5583            outer_first = 'r'
5584            last = 'r'
5585          elseif attr_d == 2 then
5586            outer_first = 'r'
5587            last = 'al'
5588          else
5589            outer_first = 'l'
5590            last = 'l'
5591          end
5592          outer = last
5593          has_en = false
5594          first_et = nil
5595          new_d = false
5596        end
5597
5598        if glue_d then
5599          if (d == 'l' and 'l' or 'r') ~= glue_d then
5600            table.insert(nodes, {glue_i, 'on', nil})
5601          end
5602          glue_d = nil
5603          glue_i = nil
5604        end
5605
```

```
5606      elseif item.id == DIR then
5607        d = nil
5608        new_d = true
5609
5610      elseif item.id == node.id'glue' and item.subtype == 13 then
5611        glue_d = d
5612        glue_i = item
5613        d = nil
5614
5615      elseif item.id == node.id'math' then
5616        inmath = (item.subtype == 0)
5617
5618      else
5619        d = nil
5620      end
5621
5622      -- AL <= EN/ET/ES     -- W2 + W3 + W6
5623      if last == 'al' and d == 'en' then
5624        d = 'an'            -- W3
5625      elseif last == 'al' and (d == 'et' or d == 'es') then
5626        d = 'on'            -- W6
5627      end
5628
5629      -- EN + CS/ES + EN      -- W4
5630      if d == 'en' and #nodes >= 2 then
5631        if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
5632            and nodes[#nodes-1][2] == 'en' then
5633          nodes[#nodes][2] = 'en'
5634        end
5635      end
5636
5637      -- AN + CS + AN         -- W4 too, because uax9 mixes both cases
5638      if d == 'an' and #nodes >= 2 then
5639        if (nodes[#nodes][2] == 'cs')
5640            and nodes[#nodes-1][2] == 'an' then
5641          nodes[#nodes][2] = 'an'
5642        end
5643      end
5644
5645      -- ET/EN                -- W5 + W7->l / W6->on
5646      if d == 'et' then
5647        first_et = first_et or (#nodes + 1)
5648      elseif d == 'en' then
5649        has_en = true
5650        first_et = first_et or (#nodes + 1)
5651      elseif first_et then        -- d may be nil here !
5652        if has_en then
5653          if last == 'l' then
5654            temp = 'l'     -- W7
5655          else
5656            temp = 'en'    -- W5
5657          end
5658        else
5659          temp = 'on'      -- W6
5660        end
5661        for e = first_et, #nodes do
5662          if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
5663        end
5664        first_et = nil
```

```
5665        has_en = false
5666      end
5667
5668      if d then
5669        if d == 'al' then
5670          d = 'r'
5671          last = 'al'
5672        elseif d == 'l' or d == 'r' then
5673          last = d
5674        end
5675        prev_d = d
5676        table.insert(nodes, {item, d, outer_first})
5677      end
5678
5679      outer_first = nil
5680
5681  end
5682
5683  -- TODO -- repeated here in case EN/ET is the last node. Find a
5684  -- better way of doing things:
5685  if first_et then        -- dir may be nil here !
5686    if has_en then
5687      if last == 'l' then
5688        temp = 'l'     -- W7
5689      else
5690        temp = 'en'    -- W5
5691      end
5692    else
5693      temp = 'on'      -- W6
5694    end
5695    for e = first_et, #nodes do
5696      if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
5697    end
5698  end
5699
5700  -- dummy node, to close things
5701  table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
5702
5703  --------------  NEUTRAL -----------------
5704
5705  outer = save_outer
5706  last = outer
5707
5708  local first_on = nil
5709
5710  for q = 1, #nodes do
5711    local item
5712
5713    local outer_first = nodes[q][3]
5714    outer = outer_first or outer
5715    last = outer_first or last
5716
5717    local d = nodes[q][2]
5718    if d == 'an' or d == 'en' then d = 'r' end
5719    if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
5720
5721    if d == 'on' then
5722      first_on = first_on or q
5723    elseif first_on then
```

```
5724      if last == d then
5725        temp = d
5726      else
5727        temp = outer
5728      end
5729      for r = first_on, q - 1 do
5730        nodes[r][2] = temp
5731        item = nodes[r][1]    -- MIRRORING
5732        if Babel.mirroring_enabled and item.id == GLYPH
5733            and temp == 'r' and characters[item.char] then
5734          local font_mode = font.fonts[item.font].properties.mode
5735          if font_mode ~= 'harf' and font_mode ~= 'plug' then
5736            item.char = characters[item.char].m or item.char
5737          end
5738        end
5739      end
5740      first_on = nil
5741    end
5742
5743    if d == 'r' or d == 'l' then last = d end
5744  end
5745
5746  --------------  IMPLICIT, REORDER ----------------
5747
5748  outer = save_outer
5749  last = outer
5750
5751  local state = {}
5752  state.has_r = false
5753
5754  for q = 1, #nodes do
5755
5756    local item = nodes[q][1]
5757
5758    outer = nodes[q][3] or outer
5759
5760    local d = nodes[q][2]
5761
5762    if d == 'nsm' then d = last end              -- W1
5763    if d == 'en' then d = 'an' end
5764    local isdir = (d == 'r' or d == 'l')
5765
5766    if outer == 'l' and d == 'an' then
5767      state.san = state.san or item
5768      state.ean = item
5769    elseif state.san then
5770      head, state = insert_numeric(head, state)
5771    end
5772
5773    if outer == 'l' then
5774      if d == 'an' or d == 'r' then     -- im -> implicit
5775        if d == 'r' then state.has_r = true end
5776        state.sim = state.sim or item
5777        state.eim = item
5778      elseif d == 'l' and state.sim and state.has_r then
5779        head, state = insert_implicit(head, state, outer)
5780      elseif d == 'l' then
5781        state.sim, state.eim, state.has_r = nil, nil, false
5782      end
```

```
5783       else
5784         if d == 'an' or d == 'l' then
5785           if nodes[q][3] then -- nil except after an explicit dir
5786             state.sim = item  -- so we move sim 'inside' the group
5787           else
5788             state.sim = state.sim or item
5789           end
5790           state.eim = item
5791         elseif d == 'r' and state.sim then
5792           head, state = insert_implicit(head, state, outer)
5793         elseif d == 'r' then
5794           state.sim, state.eim = nil, nil
5795         end
5796       end
5797
5798       if isdir then
5799         last = d            -- Don't search back - best save now
5800       elseif d == 'on' and state.san  then
5801         state.san = state.san or item
5802         state.ean = item
5803       end
5804
5805   end
5806
5807   return node.prev(head) or head
5808 end
5809 ⟨/basic⟩
```

14 Data for CJK

It is a boring file and it is not shown here (see the generated file), but here is a sample:

```
[0x0021]={c='ex'},
[0x0024]={c='pr'},
[0x0025]={c='po'},
[0x0028]={c='op'},
[0x0029]={c='cp'},
[0x002B]={c='pr'},
```

For the meaning of these codes, see the Unicode standard.

15 The 'nil' language

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

```
5810 ⟨*nil⟩
5811 \ProvidesLanguage{nil}[⟨⟨date⟩⟩ ⟨⟨version⟩⟩ Nil language]
5812 \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.

```
5813 \ifx\l@nil\@undefined
5814   \newlanguage\l@nil
```

```
5815    \@namedef{bbl@hyphendata@\the\l@nil}{{}{}}% Remove warning
5816    \let\bbl@elt\relax
5817    \edef\bbl@languages{%  Add it to the list of languages
5818      \bbl@languages\bbl@elt{nil}{\the\l@nil}{}{}}
5819 \fi
```

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

```
5820 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}
```

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

\captionnil
\datenil
```
5821 \let\captionsnil\@empty
5822 \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.

```
5823 \ldf@finish{nil}
5824 ⟨/nil⟩
```

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

16.1   Not renaming `hyphen.tex`

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

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

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

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

```
5825 ⟨*bplain | blplain⟩
5826 \catcode`\{=1 % left brace is begin-group character
5827 \catcode`\}=2 % right brace is end-group character
5828 \catcode`\#=6 % hash mark is macro parameter character
```

If a file called hyphen.cfg can be found, we make sure that *it* will be read instead of the file hyphen.tex. We do this by first saving the original meaning of \input (and I use a one letter control sequence for that so as not to waste multi-letter control sequence on this in the format).

```
5829 \openin 0 hyphen.cfg
5830 \ifeof0
5831 \else
5832   \let\a\input
```

Then \input is defined to forget about its argument and load hyphen.cfg instead. Once that's done the original meaning of \input can be restored and the definition of \a can be forgotten.

```
5833  \def\input #1 {%
5834    \let\input\a
5835    \a hyphen.cfg
5836    \let\a\undefined
5837  }
5838 \fi
5839 ⟨/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.

```
5840 ⟨bplain⟩\a plain.tex
5841 ⟨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.

```
5842 ⟨bplain⟩\def\fmtname{babel-plain}
5843 ⟨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.

```
5844 ⟨⟨∗Emulate LaTeX⟩⟩ ≡
5845   % == Code for plain ==
5846 \def\@empty{}
5847 \def\loadlocalcfg#1{%
5848   \openin0#1.cfg
5849   \ifeof0
5850     \closein0
5851   \else
5852     \closein0
5853     {\immediate\write16{**********************************}%
5854     \immediate\write16{* Local config file #1.cfg used}%
5855     \immediate\write16{*}%
5856     }
5857     \input #1.cfg\relax
5858   \fi
5859   \@endofldf}
```

16.3   General tools

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

```
5860 \long\def\@firstofone#1{#1}
5861 \long\def\@firstoftwo#1#2{#1}
5862 \long\def\@secondoftwo#1#2{#2}
5863 \def\@nnil{\@nil}
5864 \def\@gobbletwo#1#2{}
5865 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
5866 \def\@star@or@long#1{%
5867   \@ifstar
5868   {\let\l@ngrel@x\relax#1}%
5869   {\let\l@ngrel@x\long#1}}
```

```
5870 \let\l@ngrel@x\relax
5871 \def\@car#1#2\@nil{#1}
5872 \def\@cdr#1#2\@nil{#2}
5873 \let\@typeset@protect\relax
5874 \let\protected@edef\edef
5875 \long\def\@gobble#1{}
5876 \edef\@backslashchar{\expandafter\@gobble\string\\}
5877 \def\strip@prefix#1>{}
5878 \def\g@addto@macro#1#2{{%
5879     \toks@\expandafter{#1#2}%
5880     \xdef#1{\the\toks@}}}
5881 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
5882 \def\@nameuse#1{\csname #1\endcsname}
5883 \def\@ifundefined#1{%
5884   \expandafter\ifx\csname#1\endcsname\relax
5885     \expandafter\@firstoftwo
5886   \else
5887     \expandafter\@secondoftwo
5888   \fi}
5889 \def\@expandtwoargs#1#2#3{%
5890   \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a
5891 \def\zap@space#1 #2{%
5892   #1%
5893   \ifx#2\@empty\else\expandafter\zap@space\fi
5894   #2}
5895 \let\bbl@trace\@gobble
```

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

```
5896 \ifx\@preamblecmds\@undefined
5897   \def\@preamblecmds{}
5898 \fi
5899 \def\@onlypreamble#1{%
5900   \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
5901     \@preamblecmds\do#1}}
5902 \@onlypreamble\@onlypreamble
```

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

```
5903 \def\begindocument{%
5904   \@begindocumenthook
5905   \global\let\@begindocumenthook\@undefined
5906   \def\do##1{\global\let##1\@undefined}%
5907   \@preamblecmds
5908   \global\let\do\noexpand}
5909 \ifx\@begindocumenthook\@undefined
5910   \def\@begindocumenthook{}
5911 \fi
5912 \@onlypreamble\@begindocumenthook
5913 \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.

```
5914 \def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
5915 \@onlypreamble\AtEndOfPackage
5916 \def\@endofldf{}
5917 \@onlypreamble\@endofldf
5918 \let\bbl@afterlang\@empty
5919 \chardef\bbl@opt@hyphenmap\z@
```

LaTeX needs to be able to switch off writing to its auxiliary files; plain doesn't have them by default. There is a trick to hide some conditional commands from the outer \ifx. The same trick is applied below.

```
5920 \catcode`\&=\z@
5921 \ifx&if@filesw\@undefined
5922   \expandafter\let\csname if@filesw\expandafter\endcsname
5923     \csname iffalse\endcsname
5924 \fi
5925 \catcode`\&=4
```

Mimick LaTeX's commands to define control sequences.

```
5926 \def\newcommand{\@star@or@long\new@command}
5927 \def\new@command#1{%
5928   \@testopt{\@newcommand#1}0}
5929 \def\@newcommand#1[#2]{%
5930   \@ifnextchar [{\@xargdef#1[#2]}%
5931                 {\@argdef#1[#2]}}
5932 \long\def\@argdef#1[#2]#3{%
5933   \@yargdef#1\@ne{#2}{#3}}
5934 \long\def\@xargdef#1[#2][#3]#4{%
5935   \expandafter\def\expandafter#1\expandafter{%
5936     \expandafter\@protected@testopt\expandafter #1%
5937     \csname\string#1\expandafter\endcsname{#3}}%
5938   \expandafter\@yargdef \csname\string#1\endcsname
5939   \tw@{#2}{#4}}
5940 \long\def\@yargdef#1#2#3{%
5941   \@tempcnta#3\relax
5942   \advance \@tempcnta \@ne
5943   \let\@hash@\relax
5944   \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
5945   \@tempcntb #2%
5946   \@whilenum\@tempcntb <\@tempcnta
5947   \do{%
5948     \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
5949     \advance\@tempcntb \@ne}%
5950   \let\@hash@##%
5951   \l@ngrel@x\expandafter\def\expandafter#1\reserved@a}
5952 \def\providecommand{\@star@or@long\provide@command}
5953 \def\provide@command#1{%
5954   \begingroup
5955     \escapechar\m@ne\xdef\@gtempa{{\string#1}}%
5956   \endgroup
5957   \expandafter\@ifundefined\@gtempa
5958     {\def\reserved@a{\new@command#1}}%
5959     {\let\reserved@a\relax
5960      \def\reserved@a{\new@command\reserved@a}}%
5961   \reserved@a}%

5962 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
5963 \def\declare@robustcommand#1{%
5964   \edef\reserved@a{\string#1}%
5965   \def\reserved@b{#1}%
5966   \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
5967   \edef#1{%
5968     \ifx\reserved@a\reserved@b
5969       \noexpand\x@protect
5970       \noexpand#1%
5971     \fi
5972     \noexpand\protect
```

```
5973        \expandafter\noexpand\csname
5974            \expandafter\@gobble\string#1 \endcsname
5975      }%
5976      \expandafter\new@command\csname
5977          \expandafter\@gobble\string#1 \endcsname
5978 }
5979 \def\x@protect#1{%
5980      \ifx\protect\@typeset@protect\else
5981          \@x@protect#1%
5982      \fi
5983 }
5984 \catcode`\&=\z@  % Trick to hide conditionals
5985    \def\@x@protect#1&fi#2#3{&fi\protect#1}
```

The following little macro `\in@` is taken from `latex.ltx`; it checks whether its first argument is part of its second argument. It uses the boolean `\in@`; allocating a new boolean inside conditionally executed code is not possible, hence the construct with the temporary definition of `\bbl@tempa`.

```
5986    \def\bbl@tempa{\csname newif\endcsname&ifin@}
5987 \catcode`\&=4
5988 \ifx\in@\@undefined
5989    \def\in@#1#2{%
5990      \def\in@@##1#1##2##3\in@@{%
5991          \ifx\in@##2\in@false\else\in@true\fi}%
5992      \in@@#2#1\in@\in@@}
5993 \else
5994    \let\bbl@tempa\@empty
5995 \fi
5996 \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).

```
5997 \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.

```
5998 \def\@ifl@aded#1#2#3#4{}
```

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

```
5999 \ifx\@tempcnta\@undefined
6000    \csname newcount\endcsname\@tempcnta\relax
6001 \fi
6002 \ifx\@tempcntb\@undefined
6003    \csname newcount\endcsname\@tempcntb\relax
6004 \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`).

```
6005 \ifx\bye\@undefined
6006    \advance\count10 by -2\relax
6007 \fi
6008 \ifx\@ifnextchar\@undefined
```

```
6009  \def\@ifnextchar#1#2#3{%
6010    \let\reserved@d=#1%
6011    \def\reserved@a{#2}\def\reserved@b{#3}%
6012    \futurelet\@let@token\@ifnch}
6013  \def\@ifnch{%
6014    \ifx\@let@token\@sptoken
6015      \let\reserved@c\@xifnch
6016    \else
6017      \ifx\@let@token\reserved@d
6018        \let\reserved@c\reserved@a
6019      \else
6020        \let\reserved@c\reserved@b
6021      \fi
6022    \fi
6023    \reserved@c}
6024  \def\:{\let\@sptoken= } \:  % this makes \@sptoken a space token
6025  \def\:{\@xifnch} \expandafter\def\: {\futurelet\@let@token\@ifnch}
6026  \fi
6027  \def\@testopt#1#2{%
6028    \@ifnextchar[{#1}{#1[#2]}}
6029  \def\@protected@testopt#1{%
6030    \ifx\protect\@typeset@protect
6031      \expandafter\@testopt
6032    \else
6033      \@x@protect#1%
6034    \fi}
6035  \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
6036        #2\relax}\fi}
6037  \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
6038          \else\expandafter\@gobble\fi{#1}}
```

16.4   Encoding related macros

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

```
6039  \def\DeclareTextCommand{%
6040    \@dec@text@cmd\providecommand
6041  }
6042  \def\ProvideTextCommand{%
6043    \@dec@text@cmd\providecommand
6044  }
6045  \def\DeclareTextSymbol#1#2#3{%
6046    \@dec@text@cmd\chardef#1{#2}#3\relax
6047  }
6048  \def\@dec@text@cmd#1#2#3{%
6049    \expandafter\def\expandafter#2%
6050      \expandafter{%
6051        \csname#3-cmd\expandafter\endcsname
6052        \expandafter#2%
6053        \csname#3\string#2\endcsname
6054      }%
6055 %    \let\@ifdefinable\@rc@ifdefinable
6056    \expandafter#1\csname#3\string#2\endcsname
6057  }
6058  \def\@current@cmd#1{%
6059    \ifx\protect\@typeset@protect\else
6060      \noexpand#1\expandafter\@gobble
6061    \fi
6062  }
```

```
6063 \def\@changed@cmd#1#2{%
6064    \ifx\protect\@typeset@protect
6065       \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
6066          \expandafter\ifx\csname ?\string#1\endcsname\relax
6067             \expandafter\def\csname ?\string#1\endcsname{%
6068                \@changed@x@err{#1}%
6069             }%
6070          \fi
6071          \global\expandafter\let
6072             \csname\cf@encoding \string#1\expandafter\endcsname
6073             \csname ?\string#1\endcsname
6074       \fi
6075       \csname\cf@encoding\string#1%
6076          \expandafter\endcsname
6077    \else
6078       \noexpand#1%
6079    \fi
6080 }
6081 \def\@changed@x@err#1{%
6082    \errhelp{Your command will be ignored, type <return> to proceed}%
6083    \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
6084 \def\DeclareTextCommandDefault#1{%
6085    \DeclareTextCommand#1?%
6086 }
6087 \def\ProvideTextCommandDefault#1{%
6088    \ProvideTextCommand#1?%
6089 }
6090 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
6091 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
6092 \def\DeclareTextAccent#1#2#3{%
6093   \DeclareTextCommand#1{#2}[1]{\accent#3 ##1}
6094 }
6095 \def\DeclareTextCompositeCommand#1#2#3#4{%
6096    \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
6097    \edef\reserved@b{\string##1}%
6098    \edef\reserved@c{%
6099     \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
6100    \ifx\reserved@b\reserved@c
6101       \expandafter\expandafter\expandafter\ifx
6102          \expandafter\@car\reserved@a\relax\relax\@nil
6103          \@text@composite
6104       \else
6105          \edef\reserved@b##1{%
6106             \def\expandafter\noexpand
6107                \csname#2\string#1\endcsname####1{%
6108                \noexpand\@text@composite
6109                   \expandafter\noexpand\csname#2\string#1\endcsname
6110                   ####1\noexpand\@empty\noexpand\@text@composite
6111                   {##1}%
6112             }%
6113          }%
6114          \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
6115       \fi
6116       \expandafter\def\csname\expandafter\string\csname
6117          #2\endcsname\string#1-\string#3\endcsname{#4}
6118    \else
6119       \errhelp{Your command will be ignored, type <return> to proceed}%
6120       \errmessage{\string\DeclareTextCompositeCommand\space used on
6121          inappropriate command \protect#1}
```

195

```
6122        \fi
6123 }
6124 \def\@text@composite#1#2#3\@text@composite{%
6125        \expandafter\@text@composite@x
6126            \csname\string#1-\string#2\endcsname
6127 }
6128 \def\@text@composite@x#1#2{%
6129        \ifx#1\relax
6130            #2%
6131        \else
6132            #1%
6133        \fi
6134 }
6135 %
6136 \def\@strip@args#1:#2-#3\@strip@args{#2}
6137 \def\DeclareTextComposite#1#2#3#4{%
6138        \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
6139        \bgroup
6140            \lccode`\@=#4%
6141            \lowercase{%
6142        \egroup
6143            \reserved@a @%
6144        }%
6145 }
6146 %
6147 \def\UseTextSymbol#1#2{%
6148 %      \let\@curr@enc\cf@encoding
6149 %      \@use@text@encoding{#1}%
6150        #2%
6151 %      \@use@text@encoding\@curr@enc
6152 }
6153 \def\UseTextAccent#1#2#3{%
6154 %      \let\@curr@enc\cf@encoding
6155 %      \@use@text@encoding{#1}%
6156 %      #2{\@use@text@encoding\@curr@enc\selectfont#3}%
6157 %      \@use@text@encoding\@curr@enc
6158 }
6159 \def\@use@text@encoding#1{%
6160 %      \edef\f@encoding{#1}%
6161 %      \xdef\font@name{%
6162 %          \csname\curr@fontshape/\f@size\endcsname
6163 %      }%
6164 %      \pickup@font
6165 %      \font@name
6166 %      \@@enc@update
6167 }
6168 \def\DeclareTextSymbolDefault#1#2{%
6169        \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
6170 }
6171 \def\DeclareTextAccentDefault#1#2{%
6172        \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
6173 }
6174 \def\cf@encoding{OT1}
```

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

```
6175 \DeclareTextAccent{\"}{OT1}{127}
6176 \DeclareTextAccent{\'}{OT1}{19}
6177 \DeclareTextAccent{\^}{OT1}{94}
```

```
6178 \DeclareTextAccent{\`}{OT1}{18}
6179 \DeclareTextAccent{\~}{OT1}{126}
```

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

```
6180 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
6181 \DeclareTextSymbol{\textquotedblright}{OT1}{`\"}
6182 \DeclareTextSymbol{\textquoteleft}{OT1}{`\`}
6183 \DeclareTextSymbol{\textquoteright}{OT1}{`\'}
6184 \DeclareTextSymbol{\i}{OT1}{16}
6185 \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`.

```
6186 \ifx\scriptsize\@undefined
6187   \let\scriptsize\sevenrm
6188 \fi
6189   % End of code for plain
6190 ⟨⟨/Emulate LaTeX⟩⟩
```

A proxy file:

```
6191 ⟨*plain⟩
6192 \input babel.def
6193 ⟨/plain⟩
```

17   Acknowledgements

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

References

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

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

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

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

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

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

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

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

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

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

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

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