

# Babel

Version 3.56.2334  
2021/04/06

Johannes L. Braams  
Original author

Javier Bezos  
Current maintainer

Localization and  
internationalization

Unicode

T<sub>E</sub>X

pdfT<sub>E</sub>X

LuaT<sub>E</sub>X

XeT<sub>E</sub>X

# Contents

<b>I</b>	<b>User guide</b>	<b>4</b>
<b>1</b>	<b>The user interface</b>	<b>4</b>
1.1	Monolingual documents . . . . .	4
1.2	Multilingual documents . . . . .	6
1.3	Mostly monolingual documents . . . . .	8
1.4	Modifiers . . . . .	8
1.5	Troubleshooting . . . . .	9
1.6	Plain . . . . .	9
1.7	Basic language selectors . . . . .	9
1.8	Auxiliary language selectors . . . . .	10
1.9	More on selection . . . . .	11
1.10	Shorthands . . . . .	12
1.11	Package options . . . . .	16
1.12	The base option . . . . .	18
1.13	ini files . . . . .	18
1.14	Selecting fonts . . . . .	26
1.15	Modifying a language . . . . .	28
1.16	Creating a language . . . . .	30
1.17	Digits and counters . . . . .	33
1.18	Dates . . . . .	35
1.19	Accessing language info . . . . .	35
1.20	Hyphenation and line breaking . . . . .	36
1.21	Transforms . . . . .	38
1.22	Selection based on BCP 47 tags . . . . .	40
1.23	Selecting scripts . . . . .	41
1.24	Selecting directions . . . . .	42
1.25	Language attributes . . . . .	46
1.26	Hooks . . . . .	46
1.27	Languages supported by babel with ldf files . . . . .	48
1.28	Unicode character properties in luatex . . . . .	49
1.29	Tweaking some features . . . . .	49
1.30	Tips, workarounds, known issues and notes . . . . .	50
1.31	Current and future work . . . . .	51
1.32	Tentative and experimental code . . . . .	51
<b>2</b>	<b>Loading languages with language.dat</b>	<b>51</b>
2.1	Format . . . . .	52
<b>3</b>	<b>The interface between the core of babel and the language definition files</b>	<b>53</b>
3.1	Guidelines for contributed languages . . . . .	54
3.2	Basic macros . . . . .	54
3.3	Skeleton . . . . .	56
3.4	Support for active characters . . . . .	57
3.5	Support for saving macro definitions . . . . .	57
3.6	Support for extending macros . . . . .	57
3.7	Macros common to a number of languages . . . . .	58
3.8	Encoding-dependent strings . . . . .	58
<b>4</b>	<b>Changes</b>	<b>62</b>
4.1	Changes in babel version 3.9 . . . . .	62

<b>II</b>	<b>Source code</b>	<b>62</b>
<b>5</b>	<b>Identification and loading of required files</b>	<b>62</b>
<b>6</b>	<b>locale directory</b>	<b>63</b>
<b>7</b>	<b>Tools</b>	<b>63</b>
7.1	Multiple languages . . . . .	67
7.2	The Package File ( <code>\LaTeX</code> , <code>babel.sty</code> ) . . . . .	68
7.3	base . . . . .	70
7.4	Conditional loading of shorthands . . . . .	72
7.5	Cross referencing macros . . . . .	73
7.6	Marks . . . . .	76
7.7	Preventing clashes with other packages . . . . .	77
7.7.1	ifthen . . . . .	77
7.7.2	varioref . . . . .	78
7.7.3	hhline . . . . .	78
7.7.4	hyperref . . . . .	78
7.7.5	fancyhdr . . . . .	79
7.8	Encoding and fonts . . . . .	79
7.9	Basic bidi support . . . . .	81
7.10	Local Language Configuration . . . . .	86
<b>8</b>	<b>The kernel of Babel (<code>babel.def</code>, <code>common</code>)</b>	<b>90</b>
8.1	Tools . . . . .	90
<b>9</b>	<b>Multiple languages</b>	<b>91</b>
9.1	Selecting the language . . . . .	93
9.2	Errors . . . . .	102
9.3	Hooks . . . . .	104
9.4	Setting up language files . . . . .	106
9.5	Shorthands . . . . .	108
9.6	Language attributes . . . . .	117
9.7	Support for saving macro definitions . . . . .	119
9.8	Short tags . . . . .	120
9.9	Hyphens . . . . .	121
9.10	Multiencoding strings . . . . .	122
9.11	Macros common to a number of languages . . . . .	129
9.12	Making glyphs available . . . . .	129
9.12.1	Quotation marks . . . . .	129
9.12.2	Letters . . . . .	131
9.12.3	Shorthands for quotation marks . . . . .	131
9.12.4	Umlauts and tremas . . . . .	132
9.13	Layout . . . . .	134
9.14	Load engine specific macros . . . . .	134
9.15	Creating and modifying languages . . . . .	134
<b>10</b>	<b>Adjusting the Babel behavior</b>	<b>155</b>
<b>11</b>	<b>Loading hyphenation patterns</b>	<b>157</b>
<b>12</b>	<b>Font handling with <code>fontspec</code></b>	<b>161</b>

<b>13</b>	<b>Hooks for XeTeX and LuaTeX</b>	<b>166</b>
13.1	XeTeX . . . . .	166
13.2	Layout . . . . .	168
13.3	LuaTeX . . . . .	169
13.4	Southeast Asian scripts . . . . .	175
13.5	CJK line breaking . . . . .	179
13.6	Automatic fonts and ids switching . . . . .	179
13.7	Layout . . . . .	192
13.8	Auto bidi with basic and basic-r . . . . .	195
<b>14</b>	<b>Data for CJK</b>	<b>206</b>
<b>15</b>	<b>The ‘nil’ language</b>	<b>206</b>
<b>16</b>	<b>Support for Plain T<sub>E</sub>X (plain.def)</b>	<b>207</b>
16.1	Not renaming hyphen.tex . . . . .	207
16.2	Emulating some L <sup>A</sup> T <sub>E</sub> X features . . . . .	208
16.3	General tools . . . . .	208
16.4	Encoding related macros . . . . .	212
<b>17</b>	<b>Acknowledgements</b>	<b>214</b>

## Troubleshooting

Paragraph ended before \UTFviii@three@octets was complete . . . . .	5
No hyphenation patterns were preloaded for (babel) the language ‘LANG’ into the format . . . . .	6
You are loading directly a language style . . . . .	9
Unknown language ‘LANG’ . . . . .	9
Argument of \language@active@arg” has an extra } . . . . .	13
Package fontspec Warning: ‘Language ‘LANG’ not available for font ‘FONT’ with script ‘SCRIPT’ ‘Default’ language used instead’ . . . . .	28
Package babel Info: The following fonts are not babel standard families . . . . .	28

# Part I

## User guide

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

**What if I'm interested only in the latest changes?** Changes and new features with relation to version 3.8 are highlighted with `New X.XX`, and there are some notes for the latest versions in [the babel repository](#). The most recent features can be still unstable.

**Can I help?** Sure! If you are interested in the  $\TeX$  multilingual support, please join the [kadingira mail list](#). You can follow the development of `babel` in [GitHub](#) and make suggestions; feel free to fork it and make pull requests. If you are the author of a package, send to me a few test files which I'll add to mine, so that possible issues can be caught in the development phase.

**It doesn't work for me!** You can ask for help in some forums like [tex.stackexchange](#), but if you have found a bug, I strongly beg you to report it in [GitHub](#), which is much better than just complaining on an e-mail list or a web forum. Remember *warnings are not errors* by themselves, they just warn about possible problems or incompatibilities.

**How can I contribute a new language?** See section 3.1 for contributing a language.

**I only need learn the most basic features.** The first subsections (1.1-1.3) describe the traditional way of loading a language (with `ldf` files), which is usually all you need. The alternative way based on `ini` files, which complements the previous one (it does *not* replace it, although it is still necessary in some languages), is described below; go to 1.13.

**I don't like manuals. I prefer sample files.** This manual contains lots of examples and tips, but in [GitHub](#) there are many [sample files](#).

## 1 The user interface

### 1.1 Monolingual documents

In most cases, a single language is required, and then all you need in  $\LaTeX$  is to load the package using its standard mechanism for this purpose, namely, passing that language as an optional argument. In addition, you may want to set the font and input encodings. Another approach is making the language a global option in order to let other packages detect and use it. This is the standard way in  $\LaTeX$  for an option – in this case a language – to be recognized by several packages.

Many languages are compatible with `xetex` and `luatex`. With them you can use `babel` to localize the documents. When these engines are used, the Latin script is covered by default in current  $\LaTeX$  (provided the document encoding is UTF-8), because the font loader is preloaded and the font is switched to `lmroman`. Other scripts require loading `fontspec`. You may want to set the font attributes with `fontspec`, too.

**EXAMPLE** Here is a simple full example for “traditional”  $\TeX$  engines (see below for `xetex` and `luatex`). The packages `fontenc` and `inputenc` do not belong to `babel`, but they are included in the example because typically you will need them. It assumes UTF-8, the default encoding:

PDFTEX

```
\documentclass{article}

\usepackage[T1]{fontenc}

\usepackage[french]{babel}

\begin{document}

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

\end{document}
```

Now consider something like:

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

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

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

LUATEX/XETEX

```
\documentclass[russian]{article}

\usepackage{babel}

\babelfont{rm}{DejaVu Serif}

\begin{document}

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

\end{document}
```

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

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

Or the more explanatory:

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

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

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

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

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

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

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

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

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

**NOTE** Although it has been customary to recommend placing `\title`, `\author` and other elements printed by `\maketitle` after `\begin{document}`, mainly because of shorthands, it is advisable to keep them in the preamble. Currently there is no real need to use shorthands in those macros.

## 1.2 Multilingual documents

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

**EXAMPLE** In  $\LaTeX$ , the preamble of the document:

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

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

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

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

Examples of cases where main is useful are the following.

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

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

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

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

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

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

**EXAMPLE** A full bilingual document with pdfTeX follows. The main language is french, which is activated when the document begins. It assumes UTF-8:

PDFTEX

```
\documentclass{article}  
  
\usepackage[T1]{fontenc}  
  
\usepackage[english,french]{babel}  
  
\begin{document}  
  
Plus ça change, plus c'est la même chose!  
  
\selectlanguage{english}  
  
And an English paragraph, with a short text in  
\foreignlanguage{french}{français}.  
  
\end{document}
```

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

LUATEX/XETEX

```
\documentclass{article}  
  
\usepackage[vietnamese,danish]{babel}  
  
\begin{document}  
  
\prefacename{} -- \alsoname{} -- \today  
  
\selectlanguage{vietnamese}
```



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

\end{document}
```

**NOTE** Once loaded a language, you can select it with the corresponding BCP47 tag. See section 1.22 for further details.

### 1.3 Mostly monolingual documents

**New 3.39** Very often, multilingual documents consist of a main language with small pieces of text in another languages (words, idioms, short sentences). Typically, all you need is to set the line breaking rules and, perhaps, the font. In such a case, babel now does not require declaring these secondary languages explicitly, because the basic settings are loaded on the fly when the language is selected (and also when provided in the optional argument of `\babelfont`, if used.)

This is particularly useful, too, when there are short texts of this kind coming from an external source whose contents are not known on beforehand (for example, titles in a bibliography). At this regard, it is worth remembering that `\babelfont` does *not* load any font until required, so that it can be used just in case.

**EXAMPLE** A trivial document with the default font in English and Spanish, and FreeSerif in Russian is:

LUATEX/XETEX

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

\babelfont[russian]{rm}{FreeSerif}

\begin{document}

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

\end{document}
```

**NOTE** Instead of its name, you may prefer to select the language with the corresponding BCP47 tag. This alternative, however, must be activated explicitly, because a two- or tree-letter word is a valid name for a language (eg, `yi`). See section 1.22 for further details.

### 1.4 Modifiers

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

```
\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.

<sup>1</sup>No predefined “axis” for modifiers are provided because languages and their scripts have quite different needs.

## 1.5 Troubleshooting

- Loading directly sty files in L<sup>A</sup>T<sub>E</sub>X (ie, `\usepackage{⟨language⟩}`) is deprecated and you will get the error:<sup>2</sup>

```
! 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:<sup>3</sup>

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

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

## 1.6 Plain

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

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

**WARNING** Not all languages provide a sty file and some of them are not compatible with Plain.<sup>4</sup>

## 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:

<sup>2</sup>In old versions the error read “You have used an old interface to call babel”, not very helpful.

<sup>3</sup>In old versions the error read “You haven’t loaded the language LANG yet”.

<sup>4</sup>Even in the babel kernel there were some macros not compatible with plain. Hopefully these issues have been fixed.

```
\selectlanguage{german}
```

This command can be used as environment, too.

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

**New 3.43** However, if the macro name does not match any language, it will get expanded as expected.

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

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

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

**\foreignlanguage** [*<option-list>*]{*<language>*}{*<text>*}

The command `\foreignlanguage` takes two arguments; the second argument is a phrase to be typeset according to the rules of the language named in its first one.

This command (1) only switches the extra definitions and the hyphenation rules for the language, *not* the names and dates, (2) does not send information about the language to auxiliary files (i.e., the surrounding language is still in force), and (3) it works even if the language has not been set as package option (but in such a case it only sets the hyphenation patterns and a warning is shown). With the `bidi` option, it also enters in horizontal mode (this is not done always for backwards compatibility).

**New 3.44** As already said, captions and dates are not switched. However, with the optional argument you can switch them, too. So, you can write:

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

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

## 1.8 Auxiliary language selectors

**\begin{otherlanguage}** {*<language>*} ... **\end{otherlanguage}**

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

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

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

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

Spaces after the environment are ignored.

`\begin{otherlanguage*}` [*<option-list>*] {*<language>*} ... `\end{otherlanguage*}`

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

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

## 1.9 More on selection

`\babeltags` {*<tag1>* = *<language1>*, *<tag2>* = *<language2>*, ...}

**New 3.9i** In multilingual documents with many language-switches the commands above can be cumbersome. With this tool shorter names can be defined. It adds nothing really new – it is just syntactical sugar.

It defines `\text<tag1>{<text>}` to be `\foreignlanguage{<language1>}{<text>}`, and `\begin{<tag1>}` to be `\begin{otherlanguage*}{<language1>}`, and so on. Note `\<tag1>` is also allowed, but remember to set it locally inside a group.

**WARNING** There is a clear drawback to this feature, namely, the ‘prefix’ `\text...` is heavily overloaded in  $\text{\LaTeX}$  and conflicts with existing macros may arise (`\textlatin`, `\textbar`, `\textit`, `\textcolor` and many others). The same applies to environments, because `arabic` conflicts with `\arabic`. Except if there is a reason for this ‘syntactical sugar’, the best option is to stick to the default selectors or to define your own alternatives.

**EXAMPLE** With

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

you can write

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

and

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

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

**NOTE** Actually, there may be another advantage in the ‘short’ syntax `\text<tag>`, namely, it is not affected by `\MakeUppercase` (while `\foreignlanguage` is).

`\babelensure` [`include=<commands>`],`exclude=<commands>`],`fontenc=<encoding>`]{<language>}

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

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

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

```
\babelensure{polish}
```

By default only the basic captions and `\today` are redefined, but you can add further macros with the key `include` in the optional argument (without commas). Macros not to be modified are listed in `exclude`. You can also enforce a font encoding with the option `fontenc`.<sup>5</sup> 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,  $\TeX$  or `\dag`). With `ini` files (see below), captions are ensured by default.

## 1.10 Shorthands

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

**NOTE** Keep in mind the following:

1. Activated chars used for two-char shorthands cannot be followed by a closing brace `}` and the spaces following are gobbled. With one-char shorthands (eg, `:`), they are preserved.
2. If on a certain level (system, language, user, language user) there is a one-char shorthand, two-char ones starting with that char and on the same level are ignored.
3. Since they are active, a shorthand cannot contain the same character in its definition (except if deactivated with, eg, `\string`).

<sup>5</sup>With it, encoded strings may not work as expected.

**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**     $\{\langle shorthands-list \rangle\}$   
**\shorthandoff**    $\ast\{\langle shorthands-list \rangle\}$

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**    $\ast\{\langle char \rangle\}$

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*\{\langle char \rangle\}` 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**    $[\langle language \rangle, \langle language \rangle, \dots]\{\langle shorthand \rangle\}\{\langle code \rangle\}$

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{\langle lang \rangle}` to the corresponding `\extras\langle lang \rangle`, as explained below). By default, user shorthands are (re)defined.

User shorthands override language ones, which in turn override system shorthands.

Language-dependent user shorthands (new in 3.9) take precedence over “normal” user shorthands.

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

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

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

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

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

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

### `\languageshorthands` $\{\langle language \rangle\}$

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).<sup>6</sup> Note that for this to work the language should have been specified as an option when loading the babel package. For example, you can use in english the shorthands defined by ngerman with

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

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

**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` $\{\langle shorthand \rangle\}$

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:

<sup>6</sup>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.

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

For your records, here is a list of shorthands, but you must double check them, as they may change:<sup>7</sup>

**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.<sup>8</sup>

`\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*{~}}
```

<sup>7</sup>Thanks to Enrico Gregorio

<sup>8</sup>This declaration serves to nothing, but it is preserved for backward compatibility.



**WARNING** Shorthands remember somehow the original character, and the fallback value is that of the latter. So, in this example, if no shorthand is 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=**  $\langle char \rangle \langle char \rangle \dots$  | 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  $\TeX$  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  $\TeX$  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  $\epsilon\TeX$  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=**  $\langle file \rangle$

Load  $\langle file \rangle$ .`cfg` instead of the default config file `bblopts.cfg` (the file is loaded even with `noconfigs`).

<b>main=</b>	<code>&lt;language&gt;</code> 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.
<b>headfoot=</b>	<code>&lt;language&gt;</code> 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.
<b>noconfigs</b>	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.
<b>showlanguages</b>	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.
<b>nocase</b>	<b>New 3.9l</b> Language settings for uppercase and lowercase mapping (as set by <code>\SetCase</code> ) are ignored. Use only if there are incompatibilities with other packages.
<b>silent</b>	<b>New 3.9l</b> No warnings and no <i>infos</i> are written to the log file. <sup>9</sup>
<b>strings=</b>	<code>generic   unicode   encoded   &lt;label&gt;   &lt;font encoding&gt;</code> Selects the encoding of strings in languages supporting this feature. Predefined labels are <code>generic</code> (for traditional $\TeX$ , L $\text{\AA}$ CR and ASCII strings), <code>unicode</code> (for engines like xetex and luatex) and <code>encoded</code> (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 <code>\MakeUppercase</code> and the like (this feature misuses some internal $\text{\LaTeX}$ tools, so use it only as a last resort).
<b>hyphenmap=</b>	<code>off   first   select   other   other*</code> <b>New 3.9g</b> Sets the behavior of case mapping for hyphenation, provided the language defines it. <sup>10</sup> It can take the following values:  <b>off</b> deactivates this feature and no case mapping is applied; <b>first</b> sets it at the first switching commands in the current or parent scope (typically, when the aux file is first read and at <code>\begin{document}</code> }, but also the first <code>\selectlanguage</code> in the preamble), and it's the default if a single language option has been stated; <sup>11</sup> <b>select</b> sets it only at <code>\selectlanguage</code> ; <b>other</b> also sets it at <code>otherlanguage</code> ; <b>other*</b> also sets it at <code>otherlanguage*</code> as well as in heads and foots (if the option <code>headfoot</code> is used) and in auxiliary files (ie, at <code>\select@language</code> ), and it's the default if several language options have been stated. The option <code>first</code> can be regarded as an optimized version of <code>other*</code> for monolingual documents. <sup>12</sup>
<b>bidi=</b>	<code>default   basic   basic-r   bidi-l   bidi-r</code>

<sup>9</sup>You can use alternatively the package `silence`.

<sup>10</sup>Turned off in plain.

<sup>11</sup>Duplicated options count as several ones.

<sup>12</sup>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]

**New 3.14** Selects the bidi algorithm to be used in luatex and xetex. See sec. 1.24.

layout=

**New 3.16** Selects which layout elements are adapted in bidi documents. See sec. 1.24.

## 1.12 The base option

With this package option babel just loads some basic macros (those in `switch.def`), defines `\AfterBabelLanguage` and exits. It also selects the hyphenation patterns for the 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`  $\langle option-name \rangle \{ \langle code \rangle \}$

This command is currently the only provided by base. Executes  $\langle code \rangle$  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  $\langle option-name \rangle$  is the same as `\CurrentOption` (which could not be the same as the option name as set in `\usepackage!`).

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

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

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

## 1.13 ini files

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

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

Most of them set the date, and many also the captions (Unicode and LICR). They will be evolving with the time to add more features (something to keep in mind if backward compatibility is important). The following section shows how to make use of them by means of `\babelprovide`. In other words, `\babelprovide` is mainly meant for auxiliary tasks, and as alternative when the `ldf`, for some reason, does work as expected.

---

think it isn't really useful, but who knows.

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

LUATEX/XETEX

```
\documentclass{book}

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

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

\begin{document}

\tableofcontents

\chapter{სამზარეულო და სუფრის ტრადიციები}

ქართული ტრადიციული სამზარეულო ერთ-ერთი უმდიდრესია მთელ მსოფლიოში.

\end{document}
```

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

- provide=\* is the option just explained, for the main language;
- provide+=\* is the same for additional languages (the main language is still the ldf file);
- provide\*=\* is the same for all languages, ie, main and additional.

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

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

Or also:

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

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

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

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

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

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

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

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

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

```
\babelprovide[import,hyphenrules=+]{lao}
\babelpatterns[lao]{lṇ lṃ lṁ lṅ lṇ lṅ % Random
```

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

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

**Latin, Greek, Cyrillic** Combining chars with the default luatex font renderer might be wrong; on then other hand, with the Harfbuzz renderer diacritics are stacked correctly, but many hyphenations points are discarded (this bug seems related to kerning, so it depends on the font). With xetex both combining characters and hyphenation work as expected (not quite, but in most cases it works; the problem here are font clusters).

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

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

af	Afrikaans <sup>ul</sup>	en-NZ	English <sup>ul</sup>
agq	Aghem	en-US	English <sup>ul</sup>
ak	Akan	en	English <sup>ul</sup>
am	Amharic <sup>ul</sup>	eo	Esperanto <sup>ul</sup>
ar	Arabic <sup>ul</sup>	es-MX	Spanish <sup>ul</sup>
ar-DZ	Arabic <sup>ul</sup>	es	Spanish <sup>ul</sup>
ar-MA	Arabic <sup>ul</sup>	et	Estonian <sup>ul</sup>
ar-SY	Arabic <sup>ul</sup>	eu	Basque <sup>ul</sup>
as	Assamese	ewo	Ewondo
asa	Asu	fa	Persian <sup>ul</sup>
ast	Asturian <sup>ul</sup>	ff	Fulah
az-Cyrl	Azerbaijani	fi	Finnish <sup>ul</sup>
az-Latn	Azerbaijani	fil	Filipino
az	Azerbaijani <sup>ul</sup>	fo	Faroese
bas	Basaa	fr	French <sup>ul</sup>
be	Belarusian <sup>ul</sup>	fr-BE	French <sup>ul</sup>
bem	Bemba	fr-CA	French <sup>ul</sup>
bez	Bena	fr-CH	French <sup>ul</sup>
bg	Bulgarian <sup>ul</sup>	fr-LU	French <sup>ul</sup>
bm	Bambara	fur	Friulian <sup>ul</sup>
bn	Bangla <sup>ul</sup>	fy	Western Frisian
bo	Tibetan <sup>u</sup>	ga	Irish <sup>ul</sup>
brx	Bodo	gd	Scottish Gaelic <sup>ul</sup>
bs-Cyrl	Bosnian	gl	Galician <sup>ul</sup>
bs-Latn	Bosnian <sup>ul</sup>	grc	Ancient Greek <sup>ul</sup>
bs	Bosnian <sup>ul</sup>	gsw	Swiss German
ca	Catalan <sup>ul</sup>	gu	Gujarati
ce	Chechen	guz	Gusii
cgg	Chiga	gv	Manx
chr	Cherokee	ha-GH	Hausa
ckb	Central Kurdish	ha-NE	Hausa <sup>l</sup>
cop	Coptic	ha	Hausa
cs	Czech <sup>ul</sup>	haw	Hawaiian
cu	Church Slavic	he	Hebrew <sup>ul</sup>
cu-Cyrs	Church Slavic	hi	Hindi <sup>u</sup>
cu-Glag	Church Slavic	hr	Croatian <sup>ul</sup>
cy	Welsh <sup>ul</sup>	hsb	Upper Sorbian <sup>ul</sup>
da	Danish <sup>ul</sup>	hu	Hungarian <sup>ul</sup>
dav	Taita	hy	Armenian <sup>u</sup>
de-AT	German <sup>ul</sup>	ia	Interlingua <sup>ul</sup>
de-CH	German <sup>ul</sup>	id	Indonesian <sup>ul</sup>
de	German <sup>ul</sup>	ig	Igbo
dje	Zarma	ii	Sichuan Yi
dsb	Lower Sorbian <sup>ul</sup>	is	Icelandic <sup>ul</sup>
dua	Duala	it	Italian <sup>ul</sup>
dyo	Jola-Fonyi	ja	Japanese
dz	Dzongkha	jgo	Ngomba
ebu	Embu	jmc	Machame
ee	Ewe	ka	Georgian <sup>ul</sup>
el	Greek <sup>ul</sup>	kab	Kabyle
el-polyton	Polytonic Greek <sup>ul</sup>	kam	Kamba
en-AU	English <sup>ul</sup>	kde	Makonde
en-CA	English <sup>ul</sup>	kea	Kabuverdianu
en-GB	English <sup>ul</sup>	khq	Koyra Chiini

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

sw	Swahili	vai	Vai
ta	Tamil <sup>u</sup>	vi	Vietnamese <sup>ul</sup>
te	Telugu <sup>ul</sup>	vun	Vunjo
teo	Teso	wae	Walser
th	Thai <sup>ul</sup>	xog	Soga
ti	Tigrinya	yav	Yangben
tk	Turkmen <sup>ul</sup>	yi	Yiddish
to	Tongan	yo	Yoruba
tr	Turkish <sup>ul</sup>	yue	Cantonese
twq	Tasawaq	zgh	Standard Moroccan Tamazight
tzm	Central Atlas Tamazight	zh-Hans-HK	Chinese
ug	Uyghur	zh-Hans-MO	Chinese
uk	Ukrainian <sup>ul</sup>	zh-Hans-SG	Chinese
ur	Urdu <sup>ul</sup>	zh-Hans	Chinese
uz-Arab	Uzbek	zh-Hant-HK	Chinese
uz-Cyrl	Uzbek	zh-Hant-MO	Chinese
uz-Latn	Uzbek	zh-Hant	Chinese
uz	Uzbek	zh	Chinese
vai-Latn	Vai	zu	Zulu
vai-Vaii	Vai		

---

In some contexts (currently `\babel font`) an ini file may be loaded by its name. Here is the list of the names currently supported. With these languages, `\babel font` 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 `\babel provide` with a valueless `import`.

---

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



chiga	french-ch
chinese-hans-hk	french-lu
chinese-hans-mo	french-luxembourg
chinese-hans-sg	french-switzerland
chinese-hans	french
chinese-hant-hk	friulian
chinese-hant-mo	fulah
chinese-hant	galician
chinese-simplified-hongkongsarchina	ganda
chinese-simplified-macausarchina	georgian
chinese-simplified-singapore	german-at
chinese-simplified	german-austria
chinese-traditional-hongkongsarchina	german-ch
chinese-traditional-macausarchina	german-switzerland
chinese-traditional	german
chinese	greek
churchslavic	gujarati
churchslavic-cyrs	gusii
churchslavic-oldcyrillic <sup>13</sup>	hausa-gh
churchsslavic-glag	hausa-ghana
churchsslavic-glagolitic	hausa-ne
cognian	hausa-niger
cornish	hausa
croatian	hawaiian
czech	hebrew
danish	hindi
duala	hungarian
dutch	icelandic
dzongkha	igbo
embu	inarisami
english-au	indonesian
english-australia	interlingua
english-ca	irish
english-canada	italian
english-gb	japanese
english-newzealand	jolafoyi
english-nz	kabuverdianu
english-unitedkingdom	kabyle
english-unitedstates	kako
english-us	kalaallisut
english	kalenjin
esperanto	kamba
estonian	kannada
ewe	kashmiri
ewondo	kazakh
faroeese	khmer
filipino	kikuyu
finnish	kinyarwanda
french-be	konkani
french-belgium	korean
french-ca	koyraborosenni
french-canada	koyrachiini

<sup>13</sup>The name in the CLDR is Old Church Slavonic Cyrillic, but it has been shortened for practical reasons.

kwasio	ossetic
kyrgyz	pashto
lakota	persian
langi	piedmontese
lao	polish
latvian	polytonicgreek
lingala	portuguese-br
lithuanian	portuguese-brazil
lowersorbian	portuguese-portugal
lsorbian	portuguese-pt
lubakatanga	portuguese
luo	punjabi-arab
luxembourgish	punjabi-arabic
luyia	punjabi-gurmukhi
macedonian	punjabi-guru
machame	punjabi
makhuwameetto	quechua
makonde	romanian
malagasy	romansh
malay-bn	rombo
malay-brunei	rundi
malay-sg	russian
malay-singapore	rwa
malay	sakha
malayalam	samburu
maltese	samin
manx	sango
marathi	sangu
masai	sanskrit-beng
mazanderani	sanskrit-bengali
meru	sanskrit-deva
meta	sanskrit-devanagari
mexican	sanskrit-gujarati
mongolian	sanskrit-gujr
morisyen	sanskrit-kannada
mundang	sanskrit-knda
nama	sanskrit-malayalam
nepali	sanskrit-mlym
newzealand	sanskrit-telu
ngiemboon	sanskrit-telugu
ngomba	sanskrit
norsk	scottishgaelic
northernluri	sena
northernsami	serbian-cyrillic-bosniaherzegovina
northndebele	serbian-cyrillic-kosovo
norwegianbokmal	serbian-cyrillic-montenegro
norwegiannynorsk	serbian-cyrillic
nswissgerman	serbian-cyrl-ba
nuer	serbian-cyrl-me
nyankole	serbian-cyrl-xk
nynorsk	serbian-cyrl
occitan	serbian-latin-bosniaherzegovina
oriya	serbian-latin-kosovo
oromo	serbian-latin-montenegro

serbian-latin	tigrinya
serbian-latn-ba	tongan
serbian-latn-me	turkish
serbian-latn-xk	turkmen
serbian-latn	ukenglish
serbian	ukrainian
shambala	uppertsorbian
shona	urdu
sichuanyi	usenglish
sinhala	usorbian
slovak	uyghur
slovene	uzbek-arab
slovenian	uzbek-arabic
soga	uzbek-cyrillic
somali	uzbek-cyrl
spanish-mexico	uzbek-latin
spanish-mx	uzbek-latn
spanish	uzbek
standardmoroccantamazight	vai-latin
swahili	vai-latn
swedish	vai-vai
swissgerman	vai-vaii
tachelhit-latin	vai
tachelhit-latn	vietnam
tachelhit-tfng	vietnamese
tachelhit-tifinagh	vunjo
tachelhit	walser
taita	welsh
tamil	westernfrisian
tasawaq	yangben
telugu	yiddish
teso	yoruba
thai	zarma
tibetan	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=abcdefghijkl`. 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`.<sup>14</sup>

`\babelfont` [*<language-list>*] {*<font-family>*} [*<font-options>*] {*<font-name>*}

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

---

<sup>14</sup>See also the package `combofont` for a complementary approach.

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

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

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

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

Babel takes care of the font language and the font script when languages are selected (as well as the writing direction); see the recognized languages above. In most cases, you will not need *font-options*, which is the same as in `fontspec`, but you may add further key/value pairs if necessary.

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

LUATEX/XETEX

```
\documentclass{article}

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

\babelprovide[import]{hebrew}

\babelfont{rm}{FreeSerif}

\begin{document}

Svenska \foreignlanguage{hebrew}{עֵבְרִית} svenska.

\end{document}
```

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

LUATEX/XETEX

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

`\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:

LUATEX/XETEX

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

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

**NOTE** You may load fontspec explicitly. For example:

LUATEX/XETEX

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

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

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

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

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

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

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

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

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

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

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

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

## 1.15 Modifying a language

Modifying the behavior of a language (say, the chapter “caption”), is sometimes necessary, but not always trivial. In the case of caption names a specific macro is provided, because this is perhaps the most frequent change:

`\setlocalecaption`  $\{\langle language-name \rangle\}\{\langle caption-name \rangle\}\{\langle string \rangle\}$

**New 3.51** Here *caption-name* is the name as string without the trailing name. An example, which also shows caption names are often a stylistic choice, is:

```
\setlocalecaption{english}{contents}{Table of Contents}
```

This works not only with existing caption names, because it also serves to define new ones by setting the *caption-name* to the name of your choice (name will be postpended). Captions so defined or redefined behave with the ‘new way’ described in the following note.

**NOTE** There are a few alternative methods:

- With data import’ed from ini files, you can modify the values of specific keys, like:

```
\babelprovide[import, captions/listtable = Lista de tablas]{spanish}
```

(In this particular case, instead of the captions group you may need to modify the captions.licr one.)

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

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

As of 3.15, there is no need to hide spaces with % (babel removes them), but it is advisable to do so. This redefinition is not activated until the language is selected.

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

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

This redefinition is immediate.

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

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

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

Macros to be run when a language is selected can be add to `\extras<lang>`:

```
\addto\extrarussian{\mymacro}
```

There is a counterpart for code to be run when a language is unselected: `\noextras<lang>`.

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

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

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

first loads `danish.ldf`, and then redefines the captions for danish (as provided by the `ini` file) and prevents hyphenation. The rest of the language definitions are not touched. Without the optional argument it just loads some additional tools if provided by the `ini` file, like extra counters.

## 1.16 Creating a language

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

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

If the language *language-name* has not been loaded as class or package option and there are no *options*, it creates an “empty” one with some defaults in its internal structure: the hyphen rules, if not available, are set to the current ones, left and right hyphen mins are set to 2 and 3. In either case, caption, date and language system are not defined.

If no `ini` file is imported with `import`, *language-name* is still relevant because in such a case the hyphenation and like breaking rules (including those for South East Asian and CJK) are based on it as provided in the `ini` file corresponding to that name; the same applies to OpenType language and script.

Conveniently, some options allow to fill the language, and `babel` warns you about what to do if there is a missing string. Very likely you will find alerts like that in the log file:

```
Package babel Warning: \chaptername not set for 'mylang'. Please,
(babel)                define it after the language has been loaded
(babel)                (typically in the preamble) with:
(babel)                \setlocalecaption{mylang}{chapter}{..}
(babel)                Reported on input line 26.
```

In most cases, you will only need to define a few macros. Note languages loaded on the fly are not yet available in the preamble.

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

```
\usepackage[danish]{babel}
\babelprovide{arhinish}
\setlocalecaption{arhinish}{chapter}{Chapitula}
\setlocalecaption{arhinish}{refname}{Refirenke}
\renewcommand\arhinishhyphenmins{22}
```

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

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

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

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

**import=** *<language-tag>*

**New 3.13** Imports data from an ini file, including captions and date (also line breaking rules in newly defined languages). For example:

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

Unicode engines load the UTF-8 variants, while 8-bit engines load the LICR (ie, with macros like `\'` or `\ss`) ones.

**New 3.23** It may be used without a value. In such a case, the ini file set in the corresponding `babel-<language>.tex` (where `<language>` is the last argument in `\babelprovide`) is imported. See the list of recognized languages above. So, the previous example can be written:

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

There are about 250 ini files, with data taken from the ldf files and the CLDR provided by Unicode. Not all languages in the latter are complete, and therefore neither are the ini files. A few languages may show a warning about the current lack of suitability of some features.

Besides `\today`, this option defines an additional command for dates: `\<language>date`, which takes three arguments, namely, year, month and day numbers. In fact, `\today` calls `\<language>today`, which in turn calls `\<language>date{\the\year}{\the\month}{\the\day}`. **New 3.44** More convenient is usually `\localedate`, which prints the date for the current locale.

**captions=** *<language-tag>*

Loads only the strings. For example:

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

**hyphenrules=** *<language-list>*

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

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

If none of the listed hyphenrules exist, the default behavior applies. Note in this example we set `chavacano` as first option – without it, it would select `spanish` even if `chavacano` exists.

A special value is `+`, which allocates a new language (in the  $\TeX$  sense). It only makes sense as the last value (or the only one; the subsequent ones are silently ignored). It is mostly useful with `luatex`, because you can add some patterns with `\babelpatterns`, as for example:



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

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

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

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

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

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

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

Remember there is an alternative syntax for the latter:

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

**script=**  $\langle script-name \rangle$

**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=**  $\langle language-name \rangle$

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

**alph=**  $\langle counter-name \rangle$

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

**Alph=**  $\langle counter-name \rangle$

Same for `\Alph`.

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

**onchar=** ids | fonts

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

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

**intraspace=**  $\langle base \rangle \langle shrink \rangle \langle stretch \rangle$

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 scripts, like Thai, and CJK.

**intrapenalty=**  $\langle penalty \rangle$

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

**mapfont=** direction

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

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

## 1.17 Digits and counters

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

For example:

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

```

\telugudigits{1234}
\telugucounter{section}
\end{document}

```

Languages providing native digits in all or some variants are:

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

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

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

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

- `\localnumeral{<style>}{<number>}`, like `\localnumeral{abjad}{15}`
- `\localecounter{<style>}{<counter>}`, like `\localecounter{lower}{section}`
- In `\babelprovide`, as an argument to the keys `alph` and `Alph`, which redefine what `\alph` and `\Alph` print. For example:

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

The styles are:

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

**Greek** lower.modern, upper.modern, lower.ancient, upper.ancient (all with keraia)  
**Khmer** consonant  
**Korean** consonant, syllable, hanja.informal, hanja.formal, hangul.formal,  
 cjk-earthly-branch, cjk-heavenly-stem, fullwidth.lower.alpha,  
 fullwidth.upper.alpha  
**Marathi** alphabetic  
**Persian** abjad, alphabetic  
**Russian** lower, lower.full, upper, upper.full  
**Syriac** letters  
**Tamil** ancient  
**Thai** alphabetic  
**Ukrainian** lower, lower.full, upper, upper.full  
**Chinese** cjk-earthly-branch, cjk-heavenly-stem, fullwidth.lower.alpha,  
 fullwidth.upper.alpha

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

## 1.18 Dates

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

**\localedate** [ $\langle\text{calendar}=\dots, \text{variant}=\dots\rangle$ ]{ $\langle\text{year}\rangle$ }{ $\langle\text{month}\rangle$ }{ $\langle\text{day}\rangle$ }

By default the calendar is the Gregorian, but a ini files may define strings for other calendars (currently ar, ar-\*, he, fa, hi.) In the latter case, the three arguments are the year, the month, and the day in those in the corresponding calendar. They are *not* the Gregorian data to be converted (which means, say, 13 is a valid month number with calendar=hebrew).

Even with a certain calendar there may be variants. In Kurmanji the default variant prints something like *30. Çiley a Pêşîn 2019*, but with variant=iza fa it prints *31'ê Çiley a Pêşînê 2019*.

## 1.19 Accessing language info

**\language** The control sequence `\language` 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** { $\langle\text{language}\rangle$ }{ $\langle\text{true}\rangle$ }{ $\langle\text{false}\rangle$ }

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

**\localeinfo** { $\langle\text{field}\rangle$ }

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

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

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

`\getlocaleproperty` \*  $\{\langle macro \rangle\}\{\langle locale \rangle\}\{\langle property \rangle\}$

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

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

the macro `\hechap` will contain the string פרק.

If the key does not exist, the macro is set to `\relax` and an error is raised. New 3.47 With the starred version no error is raised, so that you can take your own actions with undefined properties.

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

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

`\localeid`

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

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

## 1.20 Hyphenation and line breaking

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

`\babelhyphen` \*  $\{\langle type \rangle\}$

`\babelhyphen` `*{<text>}`

**New 3.9a** It is customary to classify hyphens in two types: (1) *explicit* or *hard hyphens*, which in T<sub>E</sub>X 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 T<sub>E</sub>X 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 T<sub>E</sub>X, - 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 L<sup>A</sup>T<sub>E</sub>X: (1) the character used is that set for the current font, while in L<sup>A</sup>T<sub>E</sub>X it is hardwired to - (a typical value); (2) the hyphen to be used in fonts with a negative `\hyphenchar` is -, like in L<sup>A</sup>T<sub>E</sub>X, but it can be changed to another value by redefining `\babelnullhyphen`; (3) a break after the hyphen is forbidden if preceded by a glue >0 pt (at the beginning of a word, provided it is not immediately preceded by, say, a parenthesis).

`\babelhyphenation` [`<language>`, `<language>`, ...] `{<exceptions>}`

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

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

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

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

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

`\begin{hyphenrules}`  $\{ \langle \textit{language} \rangle \}$  ... `\end{hyphenrules}`

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

`\babelpatterns`  $[ \langle \textit{language} \rangle , \langle \textit{language} \rangle , \dots ] \{ \langle \textit{patterns} \rangle \}$

**New 3.9m** *In `luatex` only,*<sup>15</sup> 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`).

## 1.21 Transforms

Transforms (only `luatex`) provide a way to process the text on the typesetting level in several language-dependent ways, like non-standard hyphenation, special line breaking rules, script to script conversion, spacing conventions and so on.<sup>16</sup>

It currently embraces `\babelprehyphenation` and `\babelposthyphenation`, which have been available for several months.

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

<sup>15</sup>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.

<sup>16</sup>They are similar in concept, but not the same, as those in Unicode.

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

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

Arabic	transliteration.dad	Applies the transliteration system devised by Yannis Haralambous for dad (simple and $\TeX$ -friendly). Not yet complete, but sufficient for most texts.
Croatian	digraphs.ligatures	Ligatures <i>DŽ, Dž, dž, LJ, Lj, lj, NJ, Nj, nj</i> . It assumes they exist. This is not the recommended way to make these transformations (the best way is with OTF features), but it can get you out of a hurry.
Greek	diaeresis.hyphen	Removes the diaeresis above iota and upsilon if hyphenated just before. It works with the three variants.
Hindi	transliteration.hk	The Harvard-Kyoto system to romanize Devanagari.
Hungarian	digraphs.hyphen	Hyphenates the long digraphs <i>ccs, ddz, ggy, lly, nny, ssz, tty</i> and <i>zsz</i> as <i>cs-cs, dz-dz</i> , etc.

**\babelposthyphenation**  $\{\langle\textit{hyphenrules-name}\rangle\}\{\langle\textit{lua-pattern}\rangle\}\{\langle\textit{replacement}\rangle\}$

**New 3.37-3.39** With *luatex* it is now possible to define non-standard hyphenation rules, like  $f-f \rightarrow ff-f$ , 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  $([\acute{u}])$ , the replacement could be  $\{1|\acute{u}|\acute{u}\}$ , which maps  $\acute{e}$  to  $\acute{i}$ , and  $\acute{u}$  to  $\acute{u}$ , so that the diaeresis is removed.

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

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

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

**\babelprehyphenation**  $\{\langle\textit{locale-name}\rangle\}\{\langle\textit{lua-pattern}\rangle\}\{\langle\textit{replacement}\rangle\}$

**New 3.44-3.52** It is similar to the latter, but (as its name implies) applied before hyphenation, which is particularly useful in transliterations. There are other differences:



(1) the first argument is the locale instead of the name of the hyphenation patterns; (2) in the search patterns = has no special meaning, while | stands for an ordinary space; (3) in the replacement, discretionary are not accepted.

It handles glyphs and spaces.

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

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

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

**EXAMPLE** The following rule prevent the word “a” from being at the end of a line:

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

## 1.22 Selection based on BCP 47 tags

**New 3.43** The recommended way to select languages is that described at the beginning of this document. However, BCP 47 tags are becoming customary, particularly in documents (or parts of documents) generated by external sources, and therefore babel will provide a set of tools to select the locales in different situations, adapted to the particular needs of each case. Currently, babel provides autoloading of locales as described in this section. In these contexts autoloading is particularly important because we may not know on beforehand which languages will be requested.

It must be activated explicitly, because it is primarily meant for special tasks. Mapping from BCP 47 codes to locale names are not hardcoded in babel. Instead the data is taken from the ini files, which means currently about 250 tags are already recognized. Babel performs a simple lookup in the following way: *fr-Latn-FR* → *fr-Latn* → *fr-FR* → *fr*. Languages with the same resolved name are considered the same. Case is normalized before, so that *fr-latn-fr* → *fr-Latn-FR*. If a tag and a name overlap, the tag takes precedence.

Here is a minimal example:

```
\documentclass{article}

\usepackage[danish]{babel}

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

\begin{document}
```

```
Chapter in Danish: \chaptername.
```

```
\selectlanguage{de-AT}
```

```
\localedate{2020}{1}{30}
```

```
\end{document}
```

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

`autoload.bcp47` with values on and off.

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

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

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

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

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

## 1.23 Selecting scripts

Currently `babel` provides no standard interface to select scripts, because they are best selected with either `\fontencoding` (low-level) or a language name (high-level). Even the Latin script may require different encodings (ie, sets of glyphs) depending on the language, and therefore such a switch would be in a sense incomplete.<sup>17</sup> 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 it 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.<sup>18</sup>

`\ensureascii`  $\{\langle text \rangle\}$

**New 3.9i** This macro makes sure  $\langle text \rangle$  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

<sup>17</sup>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.

<sup>18</sup>But still defined for backwards compatibility.

LGR or X2 (the complete list is stored in `\BabelNonASCII`, which by default is LGR, X2, OT2, OT3, OT6, LHE, LWN, LMA, LMC, LMS, LMU, but you can modify it). So, in some sense it fixes the bug described in the previous paragraph.

If non-ASCII encodings are not loaded (or no encoding at all), it is no-op (also `\TeX` and `\LaTeX` are not redefined); otherwise, `\ensureascii` switches to the encoding at the beginning of the document if ASCII-savvy, or else the last ASCII-savvy encoding loaded. For example, if you load `LY1`, `LGR`, then it is set to `LY1`, but if you load `LY1`, `T2A` it is set to `T2A`. The symbol encodings `TS1`, `T3`, and `TS3` are not taken into account, since they are not used for “ordinary” text (they are stored in `\BabelNonText`, used in some special cases when no Latin encoding is explicitly set).

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

## 1.24 Selecting directions

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

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

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

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

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

There are some package options controlling bidi writing.

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

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

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

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

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

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

```
\documentclass{article}

\usepackage[bidi=basic]{babel}

\babelprovide[import, main]{arabic}

\babelfont{rm}{FreeSerif}

\begin{document}

    وقد عرفت شبه جزيرة العرب طيلة العصر الهليني (الارقي) بـ
    Arabia أو Aravia (بالارقية 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 \textit{fuṣḥā l-‘aṣr} (MSA) and
    \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{<subsection>}.<section>`); required in `xetex` and `pdftex` for counters in general, as well as in `luatex` with `bidi=default`; required in `luatex` for numeric footnote marks  $>9$  with `bidi=basic-r` (but *not* with `bidi=basic`); note, however, it can depend on the counter format.

With counters, `\arabic` is not only considered L text always (with `\babelsublr`, see below), but also an “isolated” block which does not interact with the surrounding chars. So, while `1.2` in R text is rendered in that order with `bidi=basic` (as a decimal number), in `\arabic{c1}.\arabic{c2}` the visual order is `c2.c1`. Of course, you may always adjust the order by changing the language, if necessary.<sup>19</sup>

**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** .

<sup>19</sup>Next on the roadmap are counters and numeral systems in general. Expect some minor readjustments.

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

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

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

**EXAMPLE** Typically, in an Arabic document you would need:

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

**\babelsublr** `{\langle lr-text \rangle}`

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 `{\langle lr-text \rangle}` in L mode if necessary. It's intended for what Unicode calls weak characters, because words are best set with the corresponding language. For this reason, there is no `rl` counterpart. Any `\babelsublr` in *explicit* L mode is ignored. However, with `bidi=basic` and *implicit* L, it first returns to R and then switches to explicit L. To clarify this point, consider, in an R context:

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

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

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

**\BabelPatchSection** `{\langle section-name \rangle}`

Mainly for bidi text, but it can be useful in other cases. `\BabelPatchSection` and the corresponding option `layout=sectioning` takes a more logical approach (at least in many cases) because it applies the global language to the section format (including the `\chaptername` in `\chapter`), while the section text is still the current language. The latter is passed to tocs and marks, too, and with `sectioning` in `layout` they both reset the “global” language to the main one, while the text uses the “local” language. With `layout=sectioning` all the standard sectioning commands are redefined (it also “isolates” the page number in heads, for a proper bidi behavior), but with this command you can set them individually if necessary (but note then tocs and marks are not touched).

**\BabelFootnote** `{\langle cmd \rangle}{\langle local-language \rangle}{\langle before \rangle}{\langle after \rangle}`

**New 3.17** Something like:

```
\BabelFootnote{\parsfootnote}{\language}\{ }\}
```

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

```
\footnote{(\foreignlanguage{\language}{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}{\language}\{ }\}%
\BabelFootnote{\localfootnote}{\language}\{ }\}%
\BabelFootnote{\mainfootnote}{\language}\{ }\}
```

(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.

## 1.25 Language attributes

### `\languageattribute`

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

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

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

## 1.26 Hooks

**New 3.9a** A hook is a piece of code to be executed at certain events. Some hooks are predefined when `luatex` and `xetex` are used.

`\AddBabelHook` [*lang*]{*name*}{*event*}{*code*}

The same name can be applied to several events. Hooks may be enabled and disabled for all defined events with `\EnableBabelHook{name}`, `\DisableBabelHook{name}`. Names containing the string `babel` are reserved (they are used, for example, by `\useshortands*` to add a hook for the event `afterextras`). **New 3.33** They may be also

applied to a specific language with the optional argument; language-specific settings are executed after global ones.

Current events are the following; in some of them you can use one to three T<sub>E</sub>X parameters (#1, #2, #3), with the meaning given:

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

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

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

**defaultcommands** Used (locally) in `\StartBabelCommands`.

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

**stopcommands** Used to reset the above, if necessary.

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

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

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

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

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

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

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

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

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

**everylanguage** (language) Executed before every language patterns are loaded.

**loadkernel** (file) By default just defines a few basic commands. It can be used to define different versions of them or to load a file.

**loadpatterns** (patterns file) Loads the patterns file. Used by `luababel.def`.

**loadexceptions** (exceptions file) Loads the exceptions file. Used by `luababel.def`.

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



## 1.27 Languages supported by babel with ldf files

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

**Afrikaans** afrikaans  
**Azerbaijani** azerbaijani  
**Basque** basque  
**Breton** breton  
**Bulgarian** bulgarian  
**Catalan** catalan  
**Croatian** croatian  
**Czech** czech  
**Danish** danish  
**Dutch** dutch  
**English** english, USenglish, american, UKenglish, british, canadian, australian, newzealand  
**Esperanto** esperanto  
**Estonian** estonian  
**Finnish** finnish  
**French** french, francais, canadien, acadian  
**Galician** galician  
**German** austrian, german, germanb, ngerman, naustrian  
**Greek** greek, polutonikogreek  
**Hebrew** hebrew  
**Icelandic** icelandic  
**Indonesian** indonesian (bahasa, indon, bahasai)  
**Interlingua** interlingua  
**Irish Gaelic** irish  
**Italian** italian  
**Latin** latin  
**Lower Sorbian** lowersorbian  
**Malay** malay, melayu (bahasam)  
**North Sami** samin  
**Norwegian** norsk, nynorsk  
**Polish** polish  
**Portuguese** portuguese, brazilian (portuges, brazil)<sup>20</sup>  
**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.

<sup>20</sup>The two last name comes from the times when they had to be shortened to 8 characters

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  $\text{\LaTeX}$ .

## 1.28 Unicode character properties in luatex

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

`\babelcharproperty`  $\{ \langle \text{char-code} \rangle \} [ \langle \text{to-char-code} \rangle ] \{ \langle \text{property} \rangle \} \{ \langle \text{value} \rangle \}$

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

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

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

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

## 1.29 Tweaking some features

`\babeladjust`  $\{ \langle \text{key-value-list} \rangle \}$

**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`).

### 1.30 Tips, workarounds, known issues and notes

- If you use the document class *book* and you use `\ref` inside the argument of `\chapter` (or just use `\ref` inside `\MakeUppercase`),  $\LaTeX$  will keep complaining about an undefined label. To prevent such problems, you can revert to using uppercase labels, you can use `\lowercase{\ref{foo}}` inside the argument of `\chapter`, or, if you will not use shorthands in labels, set the `safe` option to `none` or `bib`.
- Both `ltxdoc` and `babel` use `\AtBeginDocument` to change some catcodes, and `babel` reloads `hline` 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 `hline` (`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\extrarussian{\inputencoding{koi8-r}}
```

- 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.<sup>21</sup> So, if you write a chunk of French text with `\foreignlanguage`, the apostrophes might not be taken into account. This is a limitation of  $\TeX$ , not of `babel`. Alternatively, you may use `\usesshorthands` 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.

---

<sup>21</sup>This explains why  $\LaTeX$  assumes the lowercase mapping of T1 and does not provide a tool for multiple mappings. Unfortunately, `\savingshyphcodes` is not a solution either, because `lccodes` for hyphenation are frozen in the format and cannot be changed.

**babelbib** Multilingual bibliographies.  
**microtype** Adjusts the typesetting according to some languages (kerning and spacing).  
 Ligatures can be disabled.  
**substitutefont** Combines fonts in several encodings.  
**mkpattern** Generates hyphenation patterns.  
**tracklang** Tracks which languages have been requested.  
**ucharclasses** (xetex) Switches fonts when you switch from one Unicode block to another.  
**zhspacing** Spacing for CJK documents in xetex.

### 1.31 Current and future work

The current work is focused on the so-called complex scripts in luatex. In 8-bit engines, babel provided a basic support for bidi text as part of the style for Hebrew, but it is somewhat unsatisfactory and internally replaces some hardwired commands by other hardwired commands (generic changes would be much better). Useful additions would be, for example, time, currency, addresses and personal names.<sup>22</sup>. 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)-ből”, in Spanish an item labelled “3.<sup>o</sup>” may be referred to as either “ítem 3.<sup>o</sup>” or “3.<sup>er</sup> ítem”, and so on. An option to manage bidirectional document layout in luatex (lists, footnotes, etc.) is almost finished, but xetex required more work. Unfortunately, proper support for xetex requires patching somehow lots of macros and packages (and some issues related to `\specials` remain, like color and hyperlinks), so babel resorts to the bidi package (by Vafa Khalighi). See the babel repository for a small example (`xe-bidi`).

### 1.32 Tentative and experimental code

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

#### Options for locales loaded on the fly

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

#### Labels

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

## 2 Loading languages with `language.dat`

$\TeX$  and most engines based on it (pdf $\TeX$ , xetex,  $\epsilon$ - $\TeX$ , the main exception being luatex) require hyphenation patterns to be preloaded when a format is created (eg,  $\LaTeX$ , Xe $\LaTeX$ , pdf $\LaTeX$ ). 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).

<sup>22</sup>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.

**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).<sup>23</sup> 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).<sup>24</sup>

## 2.1 Format

In that file the person who maintains a  $\text{\TeX}$  environment has to record for which languages he has hyphenation patterns *and* in which files these are stored<sup>25</sup>. 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  $\text{\TeX}$  that the hyphenation patterns just processed have to be known under an alternative name. Here is an example:

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

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

You may also set the font encoding the patterns are intended for by following the language name by a colon and the encoding code.<sup>26</sup> For example:

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

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

A typical error when using `babel` is the following:

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

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

<sup>23</sup>This feature was added to 3.9o, but it was buggy. Both 3.9o and 3.9p are deprecated.

<sup>24</sup>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`.

<sup>25</sup>This is because different operating systems sometimes use *very* different file-naming conventions.

<sup>26</sup>This is not a new feature, but in former versions it didn't work correctly.

### 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  $\text{\TeX}$  users, so the files have to be coded so that they can be read by both  $\text{\LaTeX}$  and plain  $\text{\TeX}$ . 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  $\text{\LaTeX}$  option that is to be used. These macros and their functions are discussed below. You must define all or none for a language (or a dialect); defining, say, `\date<lang>` but not `\captions<lang>` does not raise an error but can lead to unexpected results.
- When a language definition file is loaded, it can define `\l@<lang>` to be a dialect of `\language0` when `\l@<lang>` is undefined.
- Language names must be all lowercase. If an unknown language is selected, babel will attempt setting it after lowercasing its name.
- The semantics of modifiers is not defined (on purpose). In most cases, they will just be simple separated options (eg, `spanish`), but a language might require, say, a set of options organized as a tree with suboptions (in such a case, the recommended separator is `/`).

Some recommendations:

- The preferred shorthand is `"`, which is not used in  $\text{\LaTeX}$  (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.<sup>27</sup>
- Please, for “private” internal macros do not use the `\bbl@` prefix. It is used by babel and it can lead to incompatibilities.

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

### 3.1 Guidelines for contributed languages

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

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

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

- Do not hesitate stating on the file heads you are the author and the maintainer, if you actually are. There is no need to state the babel maintainer(s) as authors if they have not contributed significantly to your language files.
- Fonts are not strictly part of a language, so they are best placed in the corresponding TeX tree. This includes not only `tfm`, `vf`, `ps1`, `otf`, `mf` files and the like, but also `fd` ones.
- Font and input encodings are usually best placed in the corresponding tree, too, but sometimes they belong more naturally to the babel style. Note you may also need to define a LICR.
- Babel ldf files may just interface a framework, as it happens often with Oriental languages/scripts. This framework is best placed in its own directory.

The following page provides a starting point for ldf files:

<http://www.texnia.com/incubator.html>. See also

<https://github.com/latex3/babel/blob/master/news-guides/guides/list-of-locale-templates.md>.

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

### 3.2 Basic macros

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

`\addlanguage` The macro `\addlanguage` is a non-outer version of the macro `\newlanguage`, defined in plain.tex version 3.x. Here “language” is used in the TeX sense of set of hyphenation patterns.

`\adddialect` The macro `\adddialect` can be used when two languages can (or must) use the same hyphenation patterns. This can also be useful for languages for which no patterns are

<sup>27</sup>But not removed, for backward compatibility.

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 T<sub>E</sub>X sense of set of hyphenation patterns. The macro `\langle lang \rangle 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.)

<code>\providehyphenmins</code>	The macro <code>\providehyphenmins</code> should be used in the language definition files to set <code>\lefthyphenmin</code> and <code>\righthyphenmin</code> . 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 <i>not</i> set them).
<code>\captions&lt;lang&gt;</code>	The macro <code>\captions&lt;lang&gt;</code> defines the macros that hold the texts to replace the original hard-wired texts.
<code>\date&lt;lang&gt;</code>	The macro <code>\date&lt;lang&gt;</code> defines <code>\today</code> .
<code>\extras&lt;lang&gt;</code>	The macro <code>\extras&lt;lang&gt;</code> 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.
<code>\noextras&lt;lang&gt;</code>	Because we want to let the user switch between languages, but we do not know what state T <sub>E</sub> X might be in after the execution of <code>\extras&lt;lang&gt;</code> , a macro that brings T <sub>E</sub> X into a predefined state is needed. It will be no surprise that the name of this macro is <code>\noextras&lt;lang&gt;</code> .
<code>\bbl@declare@ttribute</code>	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.
<code>\main@language</code>	To postpone the activation of the definitions needed for a language until the beginning of a document, all language definition files should use <code>\main@language</code> instead of <code>\selectlanguage</code> . This will just store the name of the language, and the proper language will be activated at the start of the document.
<code>\ProvidesLanguage</code>	The macro <code>\ProvidesLanguage</code> should be used to identify the language definition files. Its syntax is similar to the syntax of the L <sup>A</sup> T <sub>E</sub> X command <code>\ProvidesPackage</code> .
<code>\LdfInit</code>	The macro <code>\LdfInit</code> 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 <code>@</code> -sign, preventing the <code>.ldf</code> file from being processed twice, etc.
<code>\ldf@quit</code>	The macro <code>\ldf@quit</code> does work needed if a <code>.ldf</code> file was processed earlier. This includes resetting the category code of the <code>@</code> -sign, preparing the language to be activated at <code>\begin{document}</code> time, and ending the input stream.
<code>\ldf@finish</code>	The macro <code>\ldf@finish</code> does work needed at the end of each <code>.ldf</code> file. This includes resetting the category code of the <code>@</code> -sign, loading a local configuration file, and preparing the language to be activated at <code>\begin{document}</code> time.
<code>\loadlocalcfg</code>	After processing a language definition file, L <sup>A</sup> T <sub>E</sub> X can be instructed to load a local configuration file. This file can, for instance, be used to add strings to <code>\captions&lt;lang&gt;</code> to support local document classes. The user will be informed that this configuration file has been loaded. This macro is called by <code>\ldf@finish</code> .
<code>\substitutefontfamily</code>	(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 <code>.fd</code> file will instruct L <sup>A</sup> T <sub>E</sub> X 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>

\bbI@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\monthname{<name of first month>}
% More strings

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

\StartBabelCommands*{<dialect>}{date}
\SetString\monthname{<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
```

<code>\savebox{\myeye}{\eye}%</code>	And direct usage
<code>\newsavebox{\myeye}</code>	
<code>\newcommand\myanchor{\anchor}%</code>	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  $\TeX$  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  $\TeX$ book states: “Plain  $\TeX$  includes a macro called `\dospecials` that is essentially a set macro, representing the set of all characters that have a special category code.” [4, p. 380]

`\bbl@remove@special`

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`.  $\TeX$  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 *redefine* 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<sup>28</sup>.

`\babel@save`

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

`\babel@savevariable`

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

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

### 3.6 Support for extending macros

`\addto`

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

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

<sup>28</sup>This mechanism was introduced by Bernd Raichle.

### 3.7 Macros common to a number of languages

<code>\bbl@allowhyphens</code>	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 <code>\bbl@allowhyphens</code> can be used.
<code>\allowhyphens</code>	Same as <code>\bbl@allowhyphens</code> , but does nothing if the encoding is T1. It is intended mainly for characters provided as real glyphs by this encoding but constructed with <code>\accent</code> in OT1. Note the previous command ( <code>\bbl@allowhyphens</code> ) has different applications (hyphens and discretionary) than this one (composite chars). Note also prior to version 3.7, <code>\allowhyphens</code> had the behavior of <code>\bbl@allowhyphens</code> .
<code>\set@low@box</code>	For some languages, quotes need to be lowered to the baseline. For this purpose the macro <code>\set@low@box</code> is available. It takes one argument and puts that argument in an <code>\hbox</code> , at the baseline. The result is available in <code>\box0</code> for further processing.
<code>\save@sf@q</code>	Sometimes it is necessary to preserve the <code>\spacefactor</code> . For this purpose the macro <code>\save@sf@q</code> is available. It takes one argument, saves the current <code>spacefactor</code> , executes the argument, and restores the <code>spacefactor</code> .
<code>\bbl@frenchspacing</code> <code>\bbl@nonfrenchspacing</code>	The commands <code>\bbl@frenchspacing</code> and <code>\bbl@nonfrenchspacing</code> can be used to properly switch French spacing on and off.

### 3.8 Encoding-dependent strings

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

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

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

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

`\StartBabelCommands`  $\{ \langle \textit{language-list} \rangle \} \{ \langle \textit{category} \rangle \} [ \langle \textit{selector} \rangle ]$

The  $\langle \textit{language-list} \rangle$  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 an 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.<sup>29</sup> It may be empty, too, but in such a case using `\SetString` is an error (but not `\SetCase`).

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

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

\EndBabelCommands
```

A real example is:

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

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

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

\StartBabelCommands{german}{date}
\SetString\monthinname{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}
```

<sup>29</sup>In future releases further categories may be added.

```

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

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

\EndBabelCommands

```

When used in ldf files, previous values of  $\langle category \rangle \langle language \rangle$  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  $\langle date \rangle \langle language \rangle$  exists).

**\StartBabelCommands**  $\star \{ \langle language-list \rangle \} \{ \langle category \rangle \} [ \langle selector \rangle ]$

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.<sup>30</sup>

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

**\AfterBabelCommands**  $\{ \langle code \rangle \}$

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

**\SetString**  $\{ \langle macro-name \rangle \} \{ \langle string \rangle \}$

Adds  $\langle macro-name \rangle$  to the current category, and defines globally  $\langle lang-macro-name \rangle$  to  $\langle code \rangle$  (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**  $\{ \langle macro-name \rangle \} \{ \langle string-list \rangle \}$

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

```

\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**  $[ \langle map-list \rangle ] \{ \langle toupper-code \rangle \} \{ \langle tolower-code \rangle \}$

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

<sup>30</sup>This replaces in 3.9g a short-lived  $\backslash UseStrings$  which has been removed because it did not work.

arguments take precedence over the optional one. This command, unlike `\SetString`, is executed always (even without strings), and it is intended for minor readjustments only. For example, as T1 is the default case mapping in  $\text{\LaTeX}$ , we can set for Turkish:

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

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

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

\EndBabelCommands
```

(Note the mapping for OT1 is not complete.)

`\SetHyphenMap`  $\{ \langle to\text{-}lower\text{-}macros \rangle \}$

**New 3.9g** Case mapping serves in  $\text{\TeX}$  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  $\text{\TeX}$  primitive (`\lccode`), `babel` sets them separately. There are three helper macros to be used inside `\SetHyphenMap`:

- `\BabelLower{<uccode>}{<lccode>}` is similar to `\lccode` but it's ignored if the char has been set and saves the original `lccode` to restore it when switching the language (except with `hyphenmap=first`).
- `\BabelLowerMM{<uccode-from>}{<uccode-to>}{<step>}{<lccode-from>}` loops though the given uppercase codes, using the step, and assigns them the `lccode`, which is also increased (MM stands for *many-to-many*).
- `\BabelLowerMO{<uccode-from>}{<uccode-to>}{<step>}{<lccode>}` loops though the given uppercase codes, using the step, and assigns them the `lccode`, which is fixed (MO stands for *many-to-one*).

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

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

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

## 4 Changes

### 4.1 Changes in babel version 3.9

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

- `\select@language` did not set `\language`. This meant the language in force when auxiliary files were loaded was the one used in, for example, shorthands – if the language was german, a `\select@language{spanish}` had no effect.
- `\foreignlanguage` and `otherlanguage*` messed up `\extras<language>`. Scripts, encodings and many other things were not switched correctly.
- The `:ENC` mechanism for hyphenation patterns used the encoding of the *previous* language, not that of the language being selected.
- `'` (with `activeacute`) had the original value when writing to an auxiliary file, and things like an infinite loop can happen. It worked incorrectly with `^` (if activated) and also if deactivated.
- Active chars were not reset at the end of language options, and that led to incompatibilities between languages.
- `\textormath` raised an 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](mailto: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 two parts: a generic one and a second one only for LaTeX.

**babel.sty** is the  $\TeX$  package, which sets options and loads language styles.

**plain.def** defines some  $\TeX$  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 appropriated 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 encodings.

**[captions]** section of captions in the file charset

**[captions.licr]** same, but in pure ASCII using the LICR

**date.long** fields are as in the CLDR, but the syntax is different. Anything inside brackets is a date field (eg, MMMM for the month name) and anything outside is text. In addition, [ ] is a non breakable space and [ . ] is an abbreviation dot.

Keys may be further qualified in a particular language with a suffix starting with a uppercase letter. It can be just a letter (eg, babel.name.A, babel.name.B) or a name (eg, date.long.Nominative, date.long.Formal, but no language is currently using the latter). *Multi-letter* qualifiers are forward compatible in the sense they won’t conflict with new “global” keys (which start always with a lowercase case). There is an exception, however: the section counters has been devised to have arbitrary keys, so you can add lowercased keys if you want.

## 7 Tools

```
1 <<version=3.56.2334>>
2 <<date=2021/04/06>>
```

**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  $\TeX$  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\language\endcsname}
14 \def\bbl@loop#1#2#3{\bbl@loop#1{#3}#2,\@nnil,}
15 \def\bbl@loopx#1#2{\expandafter\bbl@loop\expandafter#1\expandafter{#2}}
16 \def\bbl@loop#1#2#3,{%
17   \ifx\@nnil#3\relax\else
18     \def#1{#3}#2\bbl@afterfi\bbl@loop#1{#2}%
19   \fi}
20 \def\bbl@for#1#2#3{\bbl@loopx#1{#2}{\ifx#1\@empty\else#3\fi}}

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

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

\bbl@afterelse Because the code that is used in the handling of active characters may need to look ahead, we take
\bbl@afterfi extra care to ‘throw’ it over the \else and \fi parts of an \if-statement31. These macros will break
if another \if... \fi statement appears in one of the arguments and it is not enclosed in braces.

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

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

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

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

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

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

```

<sup>31</sup>This code is based on code presented in TUGboat vol. 12, no2, June 1991 in “An expansion Power Lemma” by Sonja Maus.

```

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

```

**\bbl@ifblank** A tool from url, by Donald Arseneau, which tests if a string is empty or space. The companion macros tests if a macro is defined with some ‘real’ value, ie, not `\relax` and not empty,

```

68 \def\bbl@ifblank#1{%
69 \bbl@ifblank@i#1\@nil\@nil\@secondoftwo\@firstoftwo\@nil}
70 \long\def\bbl@ifblank@i#1#2\@nil#3#4#5\@nil{#4}
71 \def\bbl@ifset#1#2#3{%
72 \bbl@ifunset{#1}{#3}{\bbl@exp{\@nil\bbl@ifblank{#1}}{#3}{#2}}}

```

For each element in the comma separated `<key>=<value>` list, execute `<code>` with #1 and #2 as the key and the value of current item (trimmed). In addition, the item is passed verbatim as #3. With the `<key>` alone, it passes `\@empty` (ie, the macro thus named, not an empty argument, which is what you get with `<key>=` and no value).

```

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

```

A *for* loop. Each item (trimmed), is #1. It cannot be nested (it’s doable, but we don’t need it).

```

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

```

**\bbl@replace**

```

93 \def\bbl@replace#1#2#3{% in #1 -> repl #2 by #3
94 \toks@{}}

```

```

95 \def\bbl@replace@aux##1#2##2#2{%
96   \ifx\bbl@nil##2%
97     \toks@\expandafter{\the\toks@##1}%
98   \else
99     \toks@\expandafter{\the\toks@##1#3}%
100    \bbl@afterfi
101    \bbl@replace@aux##2#2%
102  \fi}%
103 \expandafter\bbl@replace@aux#1#2\bbl@nil#2%
104 \edef#1{\the\toks@}%

```

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

```

105 \ifx\detokenize\undefined\else % Unused macros if old Plain TeX
106   \bbl@exp{\def\\bbl@parsedef##1\detokenize{macro:}}#2->#3\relax{%
107     \def\bbl@tempa{#1}%
108     \def\bbl@tempb{#2}%
109     \def\bbl@tempe{#3}}
110   \def\bbl@sreplace#1#2#3{%
111     \begingroup
112       \expandafter\bbl@parsedef\meaning#1\relax
113       \def\bbl@tempc{#2}%
114       \edef\bbl@tempc{\expandafter\strip@prefix\meaning\bbl@tempc}%
115       \def\bbl@tempd{#3}%
116       \edef\bbl@tempd{\expandafter\strip@prefix\meaning\bbl@tempd}%
117       \bbl@xin@{\bbl@tempc}{\bbl@tempe}% If not in macro, do nothing
118       \ifin@
119         \bbl@exp{\\bbl@replace\\bbl@tempe{\bbl@tempc}{\bbl@tempd}}%
120         \def\bbl@tempc{%      Expanded an executed below as 'uplevel'
121           \\makeatletter % "internal" macros with @ are assumed
122           \\scantokens{%
123             \bbl@tempa\\@namedef{\bbl@stripslash#1}\bbl@tempb{\bbl@tempe}}%
124           \catcode64=\the\catcode64\relax}% Restore @
125       \else
126         \let\bbl@tempc\@empty % Not \relax
127       \fi
128       \bbl@exp{%      For the 'uplevel' assignments
129       \endgroup
130       \bbl@tempc}} % empty or expand to set #1 with changes
131 \fi

```

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

```

132 \def\bbl@ifsamestring#1#2{%
133   \begingroup
134     \protected@edef\bbl@tempb{#1}%
135     \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
136     \protected@edef\bbl@tempc{#2}%
137     \edef\bbl@tempc{\expandafter\strip@prefix\meaning\bbl@tempc}%
138     \ifx\bbl@tempb\bbl@tempc
139       \aftergroup\@firstoftwo
140     \else
141       \aftergroup\@secondoftwo
142     \fi

```

```

143 \endgroup}
144 \chardef\bbl@engine=%
145 \ifx\directlua\@undefined
146 \ifx\XeTeXinputencoding\@undefined
147 \z@
148 \else
149 \tw@
150 \fi
151 \else
152 \@ne
153 \fi

```

A somewhat hackish tool (hence its name) to avoid spurious spaces in some contexts.

```

154 \def\bbl@bsphack{%
155 \ifhmode
156 \hskip\z@skip
157 \def\bbl@esphack{\loop\ifdim\lastskip>\z@\unskip\repeat\unskip}%
158 \else
159 \let\bbl@esphack\@empty
160 \fi}

```

Another hackish tool, to apply case changes inside a protected macros. It's based on the internal `\let's` made by `\MakeUppercase` and `\MakeLowercase` between things like `\oe` and `\OE`.

```

161 \def\bbl@cased{%
162 \ifx\oe\OE
163 \expandafter\in@\expandafter
164 {\expandafter\OE\expandafter}\expandafter{\oe}%
165 \ifin@
166 \bbl@afterelse\expandafter\MakeUppercase
167 \else
168 \bbl@afterfi\expandafter\MakeLowercase
169 \fi
170 \else
171 \expandafter\@firstofone
172 \fi}
173 <</Basic macros>>

```

Some files identify themselves with a  $\LaTeX$  macro. The following code is placed before them to define (and then undefine) if not in  $\LaTeX$ .

```

174 <<*Make sure ProvidesFile is defined>> ≡
175 \ifx\ProvidesFile\@undefined
176 \def\ProvidesFile#1[#2 #3 #4]{%
177 \wlog{File: #1 #4 #3 <#2>}%
178 \let\ProvidesFile\@undefined}
179 \fi
180 <</Make sure ProvidesFile is defined>>

```

## 7.1 Multiple languages

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

```

181 <<*Define core switching macros>> ≡
182 \ifx\language\@undefined
183 \csname newcount\endcsname\language
184 \fi
185 <</Define core switching macros>>

```

`\last@language` Another counter is used to store the last language defined. For pre-3.0 formats an extra counter has to be allocated.

`\addlanguage` This macro was introduced for  $\TeX$  < 2. Preserved for compatibility.

```

186 <<*Define core switching macros>> ≡
187 <<*Define core switching macros>> ≡
188 \countdef\last@language=19 % TODO. why? remove?
189 \def\addlanguage{\csname newlanguage\endcsname}
190 <</Define core switching macros>>

```

Now we make sure all required files are loaded. When the command `\AtBeginDocument` doesn't exist we assume that we are dealing with a plain-based format or  $\LaTeX$  2.09. 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 and defines a few additional package options. Apart from all the language options below we also have a few options that influence the behavior of language definition files.

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

The first two options are for debugging.

```

191 <*package>
192 \NeedsTeXFormat{LaTeX2e}[2005/12/01]
193 \ProvidesPackage{babel}[\<<date>> \<<version>> The Babel package]
194 \@ifpackagewith{babel}{debug}
195   {\providecommand\bbl@trace[1]{\message{^^J[ #1 ]}}%
196    \let\bbl@debug\@firstofone
197    \ifx\directlua\@undefined\else
198      \directlua{ Babel = Babel or {}
199                Babel.debug = true }%
200    \fi}
201   {\providecommand\bbl@trace[1]{}%
202    \let\bbl@debug\@gobble
203    \ifx\directlua\@undefined\else
204      \directlua{ Babel = Babel or {}
205                Babel.debug = false }%
206    \fi}
207 <<Basic macros>>
208 % Temporarily repeat here the code for errors. TODO.
209 \def\bbl@error#1#2{%
210   \begingroup
211     \def\{\MessageBreak}%
212     \PackageError{babel}{#1}{#2}%
213   \endgroup}
214 \def\bbl@warning#1{%
215   \begingroup
216     \def\{\MessageBreak}%
217     \PackageWarning{babel}{#1}%
218   \endgroup}
219 \def\bbl@infowarn#1{%
220   \begingroup
221     \def\{\MessageBreak}%

```

```

222     \GenericWarning
223     {(babel) \@spaces\@spaces\@spaces}%
224     {Package babel Info: #1}%
225     \endgroup}
226 \def\bbl@info#1{%
227     \beginingroup
228     \def\{\MessageBreak}%
229     \PackageInfo{babel}{#1}%
230     \endgroup}
231 \def\bbl@nocaption{\protect\bbl@nocaption@i}
232 % TODO - Wrong for \today !!! Must be a separate macro.
233 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
234     \global\@namedef{#2}{\textbf{?#1?}}}%
235     \@nameuse{#2}%
236     \edef\bbl@tempa{#1}%
237     \bbl@sreplace\bbl@tempa{name}{}}%
238     \bbl@warning{%
239         \@backslashchar#1 not set for '\language'. Please,\%
240         define it after the language has been loaded\%
241         (typically in the preamble) with\%
242         \string\setlocalecaption{\language}{\bbl@tempa}{..\}%
243         Reported}}
244 \def\bbl@tentative{\protect\bbl@tentative@i}
245 \def\bbl@tentative@i#1{%
246     \bbl@warning{%
247         Some functions for '#1' are tentative.\%
248         They might not work as expected and their behavior\%
249         may change in the future.\%
250         Reported}}
251 \def\@nolanerr#1{%
252     \bbl@error
253     {You haven't defined the language #1\space yet.\%
254     Perhaps you misspelled it or your installation\%
255     is not complete}%
256     {Your command will be ignored, type <return> to proceed}}
257 \def\@nopatterns#1{%
258     \bbl@warning
259     {No hyphenation patterns were preloaded for\%
260     the language `#1' into the format.\%
261     Please, configure your TeX system to add them and\%
262     rebuild the format. Now I will use the patterns\%
263     preloaded for \bbl@nulllanguage\space instead}}
264     % End of errors
265 \@ifpackagewith{babel}{silent}
266 {\let\bbl@info\@gobble
267  \let\bbl@infowarn\@gobble
268  \let\bbl@warning\@gobble}
269 {}
270 %
271 \def\AfterBabelLanguage#1{%
272     \global\expandafter\bbl@add\csname#1.ldf-h@@k\endcsname}%

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

273 \ifx\bbl@languages\undefined\else
274     \beginingroup
275     \catcode`\^^I=12
276     \@ifpackagewith{babel}{showlanguages}{%
277         \beginingroup

```

```

278     \def\bbl@elt#1#2#3#4{\wlog{#2^^I#1^^I#3^^I#4}}%
279     \wlog{<*languages>}%
280     \bbl@languages
281     \wlog{</languages>}%
282   \endgroup{}}
283 \endgroup
284 \def\bbl@elt#1#2#3#4{%
285   \ifnum#2=\z@
286     \gdef\bbl@nulllanguage{#1}%
287     \def\bbl@elt##1##2##3##4{%
288       \fi}%
289   \bbl@languages
290 \fi%

```

### 7.3 base

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

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

```

291 \bbl@trace{Defining option 'base'}
292 \@ifpackagewith{babel}{base}{%
293   \let\bbl@onlyswitch\@empty
294   \let\bbl@provide@locale\relax
295   \input babel.def
296   \let\bbl@onlyswitch\@undefined
297   \ifx\directlua\@undefined
298     \DeclareOption*{\bbl@patterns{\CurrentOption}}%
299   \else
300     \input luababel.def
301     \DeclareOption*{\bbl@patterns@lua{\CurrentOption}}%
302   \fi
303   \DeclareOption{base}{}%
304   \DeclareOption{showlanguages}{}%
305   \ProcessOptions
306   \global\expandafter\let\csname opt@babel.sty\endcsname\relax
307   \global\expandafter\let\csname ver@babel.sty\endcsname\relax
308   \global\let\@ifl@ter@\@ifl@ter
309   \def\@ifl@ter#1#2#3#4#5{\global\let\@ifl@ter\@ifl@ter@@}%
310   \endinput}{}%
311 % \end{macrocode}
312 %
313 % \subsection{\texttt{key=value} options and other general option}
314 %
315 %   The following macros extract language modifiers, and only real
316 %   package options are kept in the option list. Modifiers are saved
317 %   and assigned to |\BabelModifiers| at |\bbl@load@language|; when
318 %   no modifiers have been given, the former is |\relax|. How
319 %   modifiers are handled are left to language styles; they can use
320 %   |\in@|, loop them with |\@for| or load |keyval|, for example.
321 %
322 %   \begin{macrocode}
323 \bbl@trace{key=value and another general options}
324 \bbl@csarg\let{tempa\expandafter}\csname opt@babel.sty\endcsname
325 \def\bbl@tempb#1.#2{% Remove trailing dot
326   #1\ifx\@empty#2\else,\bbl@afterfi\bbl@tempb#2\fi}%
327 \def\bbl@tempd#1.#2@nnil{% TODO. Refactor lists?

```

```

328 \ifx\@empty#2%
329 \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
330 \else
331 \in@{,provide,}{, #1,}%
332 \ifin@
333 \edef\bbl@tempc{%
334 \ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.\bbl@tempb#2}%
335 \else
336 \in@{=}{ #1}%
337 \ifin@
338 \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.#2}%
339 \else
340 \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
341 \bbl@csarg\edef{mod@#1}{\bbl@tempb#2}%
342 \fi
343 \fi
344 \fi}
345 \let\bbl@tempc\@empty
346 \bbl@foreach\bbl@tempa{\bbl@tempd#1.\@empty\@nnil}
347 \expandafter\let\csname opt@babel.sty\endcsname\bbl@tempc

```

The next option tells babel to leave shorthand characters active at the end of processing the package. This is *not* the default as it can cause problems with other packages, but for those who want to use the shorthand characters in the preamble of their documents this can help.

```

348 \DeclareOption{KeepShorthandsActive}{}
349 \DeclareOption{activeacute}{}
350 \DeclareOption{activegrave}{}
351 \DeclareOption{debug}{}
352 \DeclareOption{noconfigs}{}
353 \DeclareOption{showlanguages}{}
354 \DeclareOption{silent}{}
355 \DeclareOption{mono}{}
356 \DeclareOption{shorthands=off}{\bbl@tempa shorthands=\bbl@tempa}
357 \chardef\bbl@iniflag\z@
358 \DeclareOption{provide=*}{\chardef\bbl@iniflag\@ne} % main -> +1
359 \DeclareOption{provide+=*}{\chardef\bbl@iniflag\tw@} % add = 2
360 \DeclareOption{provide*=*}{\chardef\bbl@iniflag\thr@@} % add + main
361 % A separate option
362 \let\bbl@autoload@options\@empty
363 \DeclareOption{provide=@*}{\def\bbl@autoload@options{import}}
364 % Don't use. Experimental. TODO.
365 \newif\ifbbl@single
366 \DeclareOption{selectors=off}{\bbl@singletrue}
367 <<More package options>>

```

Handling of package options is done in three passes. (I [JBL] am not very happy with the idea, anyway.) The first one processes options which has been declared above or follow the syntax <key>=<value>, the second one loads the requested languages, except the main one if set with the key main, and the third one loads the latter. First, we “flag” valid keys with a nil value.

```

368 \let\bbl@opt@shorthands\@nnil
369 \let\bbl@opt@config\@nnil
370 \let\bbl@opt@main\@nnil
371 \let\bbl@opt@headfoot\@nnil
372 \let\bbl@opt@layout\@nnil

```

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

```

373 \def\bbl@tempa#1=#2\bbl@tempa{%
374 \bbl@csarg\ifx{opt@#1}\@nnil
375 \bbl@csarg\edef{opt@#1}{#2}%

```



```

376 \else
377   \bbl@error
378   {Bad option `#1=#2'. Either you have misspelled the\\%
379    key or there is a previous setting of `#1'. Valid\\%
380    keys are, among others, `shorthands', `main', `bidi',\\%
381    `strings', `config', `headfoot', `safe', `math'.}%
382   {See the manual for further details.}
383 \fi}

```

Now the option list is processed, taking into account only currently declared options (including those declared with a =), and <key>=<value> options (the former take precedence). Unrecognized options are saved in \bbl@language@opts, because they are language options.

```

384 \let\bbl@language@opts\@empty
385 \DeclareOption*{%
386   \bbl@xin@{\string=}{\CurrentOption}%
387   \ifin@
388     \expandafter\bbl@tempa\CurrentOption\bbl@tempa
389   \else
390     \bbl@add@list\bbl@language@opts{\CurrentOption}%
391   \fi}

```

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

```

392 \ProcessOptions*

```

## 7.4 Conditional loading of shorthands

If there is no shorthands=<chars>, the original babel macros are left untouched, but if there is, these macros are wrapped (in babel.def) to define only those given.

A bit of optimization: if there is no shorthands=, then \bbl@ifshorthand is always true, and it is always false if shorthands is empty. Also, some code makes sense only with shorthands=...

```

393 \bbl@trace{Conditional loading of shorthands}
394 \def\bbl@sh@string#1{%
395   \ifx#1\@empty\else
396     \ifx#1t\string~%
397     \else\ifx#1c\string,%
398     \else\string#1%
399   \fi\fi
400   \expandafter\bbl@sh@string
401 \fi}
402 \ifx\bbl@opt@shorthands\@nnil
403   \def\bbl@ifshorthand#1#2#3{#2}%
404 \else\ifx\bbl@opt@shorthands\@empty
405   \def\bbl@ifshorthand#1#2#3{#3}%
406 \else

```

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

```

407 \def\bbl@ifshorthand#1{%
408   \bbl@xin@{\string#1}{\bbl@opt@shorthands}%
409   \ifin@
410     \expandafter\@firstoftwo
411   \else
412     \expandafter\@secondoftwo
413   \fi}

```

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

```

414 \edef\bbl@opt@shorthands{%
415   \expandafter\bbl@sh@string\bbl@opt@shorthands\@empty}%

```

The following is ignored with `shorthands=off`, since it is intended to take some additional actions for certain chars.

```
416 \bbl@ifshorthand{'}%
417   {\PassOptionsToPackage{activeacute}{babel}}{}
418 \bbl@ifshorthand{`}%
419   {\PassOptionsToPackage{activegrave}{babel}}{}
420 \fi\fi
```

With `headfoot=lang` we can set the language used in heads/foots. For example, in `babel/3796` just adds `headfoot=english`. It misuses `\@resetactivechars` but seems to work.

```
421 \ifx\bbl@opt@headfoot\@nnil\else
422   \g@addto@macro\@resetactivechars{%
423     \set@typeset@protect
424     \expandafter\select@language@x\expandafter{\bbl@opt@headfoot}%
425     \let\protect\noexpand}
426 \fi
```

For the option `safe` we use a different approach – `\bbl@opt@safe` says which macros are redefined (B for bibs and R for refs). By default, both are set.

```
427 \ifx\bbl@opt@safe\@undefined
428   \def\bbl@opt@safe{BR}
429 \fi
430 \ifx\bbl@opt@main\@nnil\else
431   \edef\bbl@language@opts{%
432     \ifx\bbl@language@opts\empty\else\bbl@language@opts,\fi
433     \bbl@opt@main}
434 \fi
```

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

```
435 \bbl@trace{Defining IfBabelLayout}
436 \ifx\bbl@opt@layout\@nnil
437   \newcommand\IfBabelLayout[3]{#3}%
438 \else
439   \newcommand\IfBabelLayout[1]{%
440     \@expandtwoargs\in@{.#1.}{.\bbl@opt@layout.}%
441     \ifin@
442       \expandafter\@firstoftwo
443     \else
444       \expandafter\@secondoftwo
445     \fi}
446 \fi
```

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

```
447 \input babel.def
```

## 7.5 Cross referencing macros

The  $\TeX$  book states:

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

When the above quote should still be true when a document is typeset in a language that has active characters, special care has to be taken of the category codes of these characters when they appear in an argument of the cross referencing macros.

When a cross referencing command processes its argument, all tokens in this argument should be character tokens with category ‘letter’ or ‘other’.

The following package options control which macros are to be redefined.

```
448 <<{*More package options}> \equiv
```

```

449 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
450 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
451 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
452 <</More package options>>

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

453 \bbl@trace{Cross referencing macros}
454 \ifx\bbl@opt@safe\@empty\else
455   \def\@newl@bel#1#2#3{%
456     {\@safe@activestrue
457       \bbl@ifunset{#1@#2}%
458         \relax
459         {\gdef\@multiplelabels{%
460           \@latex@warning@no@line{There were multiply-defined labels}}}%
461         \@latex@warning@no@line{Label `#2' multiply defined}}%
462     \global\@namedef{#1@#2}{#3}}

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

463 \CheckCommand*\@testdef[3]{%
464   \def\reserved@a{#3}%
465   \expandafter\ifx\curname#1@#2\endcurname\reserved@a
466   \else
467     \@tempwattrue
468     \fi}

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

469 \def\@testdef#1#2#3{% TODO. With @samestring?
470   \@safe@activestrue
471   \expandafter\let\expandafter\bbl@tempa\curname #1@#2\endcurname
472   \def\bbl@tempb{#3}%
473   \@safe@activesfalse
474   \ifx\bbl@tempa\relax
475   \else
476     \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
477     \fi
478     \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
479     \ifx\bbl@tempa\bbl@tempb
480     \else
481       \@tempwattrue
482       \fi}
483 \fi

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

484 \bbl@xin@{R}\bbl@opt@safe
485 \ifin@
486   \bbl@redefineroobust\ref#1{%
487     \@safe@activestrue\org@ref{#1}\@safe@activesfalse}
488   \bbl@redefineroobust\pageref#1{%
489     \@safe@activestrue\org@pageref{#1}\@safe@activesfalse}

```

```

490 \else
491   \let\org@ref\ref
492   \let\org@pageref\pageref
493 \fi

\@citex The macro used to cite from a bibliography, \cite, uses an internal macro, \@citex. It is this
internal macro that picks up the argument(s), so we redefine this internal macro and leave \cite
alone. The first argument is used for typesetting, so the shorthands need only be deactivated in the
second argument.

494 \bbl@xin@{B}\bbl@opt@safe
495 \ifin@
496   \bbl@redefine\@citex[#1]#2{%
497     \@safe@activetrue\edef\@tempa{#2}\@safe@activesfalse
498     \org@citex[#1]{\@tempa}}

Unfortunately, the packages natbib and cite need a different definition of \@citex... To begin with,
natbib has a definition for \@citex with three arguments... We only know that a package is loaded
when \begin{document} is executed, so we need to postpone the different redefinition.

499 \AtBeginDocument{%
500   \ifpackageloaded{natbib}{%

Notice that we use \def here instead of \bbl@redefine because \org@@citex is already defined and
we don't want to overwrite that definition (it would result in parameter stack overflow because of a
circular definition).
(Recent versions of natbib change dynamically \@citex, so PR4087 doesn't seem fixable in a simple
way. Just load natbib before.)

501   \def\@citex[#1][#2]#3{%
502     \@safe@activetrue\edef\@tempa{#3}\@safe@activesfalse
503     \org@citex[#1][#2]{\@tempa}}%
504   }{}}

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

505 \AtBeginDocument{%
506   \ifpackageloaded{cite}{%
507     \def\@citex[#1]#2{%
508       \@safe@activetrue\org@citex[#1]{#2}\@safe@activesfalse}%
509     }{}}

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

510 \bbl@redefine\nocite#1{%
511   \@safe@activetrue\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@activetrue
is in effect. This switch needs to be reset inside the \hbox which contains the citation label. In order
to determine during .aux file processing which definition of \bibcite is needed we define \bibcite
in such a way that it redefines itself with the proper definition. We call \bbl@cite@choice to select
the proper definition for \bibcite. This new definition is then activated.

512 \bbl@redefine\bibcite{%
513   \bbl@cite@choice
514   \bibcite}

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

515 \def\bbl@bibcite#1#2{%
516   \org@bibcite{#1}{\@safe@activesfalse#2}}

```

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

```
517 \def\bbl@cite@choice{%
518   \global\let\bbl@cite\bbl@bibcite
519   \ifpackageloaded{natbib}{\global\let\bbl@cite\org@bibcite}{}%
520   \ifpackageloaded{cite}{\global\let\bbl@cite\org@bibcite}{}%
521   \global\let\bbl@cite@choice\relax}
```

When a document is run for the first time, no .aux file is available, and `\bbl@cite` will not yet be properly defined. In this case, this has to happen before the document starts.

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

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

```
523 \bbl@redefine\@bibitem#1{%
524   \@safe@activetrue\org@bibitem{#1}\@safe@activesfalse}
525 \else
526   \let\org@nocite\nocite
527   \let\org@citex\citex
528   \let\org@bibcite\bbl@bibcite
529   \let\org@bibitem\@bibitem
530 \fi
```

## 7.6 Marks

`\markright` Because the output routine is asynchronous, we must pass the current language attribute to the head lines. To achieve this we need to adapt the definition of `\markright` and `\markboth` somewhat. However, headlines and footlines can contain text outside marks; for that we must take some actions in the output routine if the 'headfoot' options is used. We need to make some redefinitions to the output routine to avoid an endless loop and to correctly handle the page number in bidi documents.

```
531 \bbl@trace{Marks}
532 \IfBabelLayout{sectioning}
533   {\ifx\bbl@opt@headfoot\@nnil
534     \g@addto@macro\@resetactivechars{%
535       \set@typeset@protect
536       \expandafter\select@language@x\expandafter{\bbl@main@language}%
537       \let\protect\noexpand
538       \ifcase\bbl@bidimode\else % Only with bidi. See also above
539         \edef\thepage{%
540           \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
541       \fi}%
542   \fi}
543 {\ifbbl@single\else
544   \bbl@ifunset{markright }{\bbl@redefine\bbl@redefineroobust
545     \markright#1{%
546       \bbl@ifblank{#1}%
547       {\org@markright{}}}%
548       {\toks@{#1}%
549         \bbl@exp{%
550           \org@markright{\protect\foreignlanguage{\language}\thepage}%
551           {\protect\bbl@restore@actives\the\toks@}}}%
552     }%
553   }
```

`\markboth` The definition of `\markboth` is equivalent to that of `\markright`, except that we need two 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 need to do that again with the new definition of `\markboth`. (As of Oct 2019,  $\TeX$  stores the definition in an intermediate macro, so it's not necessary anymore, but it's preserved for older versions.)

```

552 \ifx\@mkboth\markboth
553 \def\bbl@tempc{\let\@mkboth\markboth}
554 \else
555 \def\bbl@tempc{}
556 \fi
557 \bbl@ifunset{markboth }\bbl@redefine\bbl@redefineroobust
558 \markboth#1#2{%
559 \protected@edef\bbl@tempb##1{%
560 \protect\foreignlanguage
561 {\language}\protect\bbl@restore@actives##1}}%
562 \bbl@ifblank{#1}%
563 {\toks@{}}%
564 {\toks@\expandafter{\bbl@tempb{#1}}}%
565 \bbl@ifblank{#2}%
566 {\@temptokena{}}%
567 {\@temptokena\expandafter{\bbl@tempb{#2}}}%
568 \bbl@exp{\@org@markboth{\the\toks@}{\the\@temptokena}}
569 \bbl@tempc
570 \fi} % end ifbbl@single, end \IfBabelLayout

```

## 7.7 Preventing clashes with other packages

### 7.7.1 ifthen

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

```

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

```

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

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

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

```

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

```

```

590         #3}%
591     }%
592 }{}%
593 }

```

### 7.7.2 varioref

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

```

594 \AtBeginDocument{%
595   \ifpackageloaded{varioref}{%
596     \bbl@redefine\@@vpageref#1[#2]#3{%
597       \@safe@activestrue
598       \org@@@vpageref{#1}[#2]{#3}%
599       \@safe@activesfalse}%
600     \bbl@redefine\vrefpagenum#1#2{%
601       \@safe@activestrue
602       \org@vrefpagenum{#1}{#2}%
603       \@safe@activesfalse}%

```

The package `varioref` defines `\Ref` to be a robust command which uppercases the first character of the reference text. In order to be able to do that it needs to access the expandable form of `\ref`. So we employ a little trick here. We redefine the (internal) command `\Ref` to call `\org@ref` instead of `\ref`. The disadvantage of this solution is that whenever the definition of `\Ref` changes, this definition needs to be updated as well.

```

604   \expandafter\def\csname Ref\endcsname#1{%
605     \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
606   }{}%
607 }
608 \fi

```

### 7.7.3 hhline

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

```

609 \AtEndOfPackage{%
610   \AtBeginDocument{%
611     \ifpackageloaded{hhline}%
612     {\expandafter\ifx\csname normal@char\string\endcsname\relax
613       \else
614         \makeatletter
615         \def\@currname{hhline}\input{hhline.sty}\makeatother
616       \fi}%
617     {}}}

```

### 7.7.4 hyperref

`\pdfstringdefDisableCommands` A number of interworking problems between `babel` and `hyperref` are tackled by `hyperref` itself. The following code was introduced to prevent some annoying warnings but it broke bookmarks. This was quickly fixed in `hyperref`, which essentially made it no-op. However, it will not be removed for the moment because `hyperref` is expecting it. TODO. Still true? Commented out in 2020/07/27.

```

618 % \AtBeginDocument{%
619 %   \ifx\pdfstringdefDisableCommands\@undefined\else
620 %     \pdfstringdefDisableCommands{\languageshorthands{system}}%
621 %   \fi}

```

### 7.7.5 fancyhdr

`\FOREIGNLANGUAGE` The package `fancyhdr` treats the running head and foot lines somewhat differently as the standard classes. A symptom of this is that the command `\foreignlanguage` which `babel` adds to the marks can end up inside the argument of `\MakeUppercase`. To prevent unexpected results we need to define `\FOREIGNLANGUAGE` here.

```
622 \DeclareRobustCommand{\FOREIGNLANGUAGE}[1]{%
623   \lowercase{\foreignlanguage{#1}}}
```

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

```
624 \def\substitutefontfamily#1#2#3{%
625   \lowercase{\immediate\openout15=#1#2.fd\relax}%
626   \immediate\write15{%
627     \string\ProvidesFile{#1#2.fd}%
628     [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
629     \space generated font description file]^^J
630     \string\DeclareFontFamily{#1}{#2}{^^J
631     \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{^^J
632     \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{^^J
633     \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{^^J
634     \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{^^J
635     \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{^^J
636     \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{^^J
637     \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{^^J
638     \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{^^J
639   }%
640   \closeout15
641 }
642 \@onlypreamble\substitutefontfamily
```

## 7.8 Encoding and fonts

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

`\ensureascii`

```
643 \bbl@trace{Encoding and fonts}
644 \newcommand\BabelNonASCII{LGR,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU,PU,PD1}
645 \newcommand\BabelNonText{TS1,T3,TS3}
646 \let\org@TeX\TeX
647 \let\org@LaTeX\LaTeX
648 \let\ensureascii\@firstofone
649 \AtBeginDocument{%
650   \in@false
651   \bbl@foreach\BabelNonASCII{% is there a text non-ascii enc?
652     \ifin@false
653       \lowercase{\bbl@xin@{,#1enc.def},{,\@filelist,}}%
654     \fi}%
655   \ifin@ % if a text non-ascii has been loaded
656     \def\ensureascii#1{{\fontencoding{OT1}\selectfont#1}}%
657     \DeclareTextCommandDefault{\TeX}{\org@TeX}%
658     \DeclareTextCommandDefault{\LaTeX}{\org@LaTeX}%
```



```

659 \def\bbl@tempb#1\@{\uppercase{\bbl@tempc#1}ENC.DEF\@empty\@}%
660 \def\bbl@tempc#1ENC.DEF#2\@{\%
661 \ifx\@empty#2\else
662 \bbl@ifunset{T@#1}%
663 }}%
664 {\bbl@xin@{,#1,},{,\BabelNonASCII,\BabelNonText,}%
665 \ifin@
666 \DeclareTextCommand{\TeX}{#1}{\ensureascii{\org@TeX}}%
667 \DeclareTextCommand{\LaTeX}{#1}{\ensureascii{\org@LaTeX}}%
668 \else
669 \def\ensureascii##1{{\fontencoding{#1}\selectfont##1}}%
670 \fi}%
671 \fi}%
672 \bbl@foreach\@filelist{\bbl@tempb#1\@}% TODO - \@@ de mas??
673 \bbl@xin@{,\cf@encoding,},{,\BabelNonASCII,\BabelNonText,}%
674 \ifin@else
675 \edef\ensureascii#1{{%
676 \noexpand\fontencoding{\cf@encoding}\noexpand\selectfont#1}}%
677 \fi
678 \fi}

```

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

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

```

679 \AtEndOfPackage{\edef\latinencoding{\cf@encoding}}

```

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

```

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

```

`\latintext` Then we can define the command `\latintext` which is a declarative switch to a latin font-encoding. Usage of this macro is deprecated.

```

704 \DeclareRobustCommand{\latintext}{%
705   \fontencoding{\latinencoding}\selectfont
706   \def\encodingdefault{\latinencoding}}

\textlatin This command takes an argument which is then typeset using the requested font encoding. In order
to avoid many encoding switches it operates in a local scope.

707 \ifx\@undefined\DeclareTextFontCommand
708   \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}
709 \else
710   \DeclareTextFontCommand{\textlatin}{\latintext}
711 \fi

```

## 7.9 Basic bidi support

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

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

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

- `pdftex` provides a minimal support for bidi text, and it must be done by hand. Vertical typesetting is not possible.
- `xetex` is somewhat better, thanks to its font engine (even if not always reliable) and a few additional tools. However, very little is done at the paragraph level. Another challenging problem is text direction does not honour  $\TeX$  grouping.
- `luatex` can provide the most complete solution, as we can manipulate almost freely the node list, the generated lines, and so on, but bidi text does not work out of the box and some development is necessary. It also provides tools to properly set left-to-right and right-to-left page layouts. As `Lua $\TeX$ -ja` shows, vertical typesetting is possible, too.

As a first step, add a handler for bidi and digits (and potentially other processes) just before `luaotfload` is applied, which is loaded by default by  $\LaTeX$ . Just in case, consider the possibility it has not been loaded.

```

712 \ifodd\bbl@engine
713   \def\bbl@activate@preotf{%
714     \let\bbl@activate@preotf\relax % only once
715     \directlua{
716       Babel = Babel or {}
717       %
718       function Babel.pre_otfload_v(head)
719         if Babel.numbers and Babel.digits_mapped then
720           head = Babel.numbers(head)
721         end
722         if Babel.bidi_enabled then
723           head = Babel.bidi(head, false, dir)
724         end
725         return head
726       end
727       %
728       function Babel.pre_otfload_h(head, gc, sz, pt, dir)
729         if Babel.numbers and Babel.digits_mapped then
730           head = Babel.numbers(head)
731         end

```

```

732     if Babel.bidi_enabled then
733         head = Babel.bidi(head, false, dir)
734     end
735     return head
736 end
737 %
738 luatexbase.add_to_callback('pre_linebreak_filter',
739     Babel.pre_otfload_v,
740     'Babel.pre_otfload_v',
741     luatexbase.priority_in_callback('pre_linebreak_filter',
742         'luaotfload.node_processor') or nil)
743 %
744 luatexbase.add_to_callback('hpack_filter',
745     Babel.pre_otfload_h,
746     'Babel.pre_otfload_h',
747     luatexbase.priority_in_callback('hpack_filter',
748         'luaotfload.node_processor') or nil)
749 }}
750 \fi

```

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

```

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

```

```

788      \bbl@loadfontspec % bidi needs fontspec
789      \fi
790      \usepackage#1{bidi}}%
791    \fi}
792  \ifnum\bbl@bidimode>200
793    \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
794      \bbl@tentative{bidi=bidi}
795      \bbl@loadxebidi{}
796    \or
797      \bbl@loadxebidi{[rldocument]}
798    \or
799      \bbl@loadxebidi{}
800    \fi
801  \fi
802 \fi
803 \ifnum\bbl@bidimode=\@ne
804   \let\bbl@beforeforeign\leavevmode
805   \ifodd\bbl@engine
806     \newattribute\bbl@attr@dir
807     \bbl@exp{\output{\bodydir\pagedir\the\output}}%
808   \fi
809   \AtEndOfPackage{%
810     \EnableBabelHook{babel-bidi}%
811     \ifodd\bbl@engine\else
812       \bbl@xebidipar
813     \fi}
814 \fi

```

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

```

815 \bbl@trace{Macros to switch the text direction}
816 \def\bbl@alscripts{,Arabic,Syriac,Thaana,}
817 \def\bbl@rscripts{% TODO. Base on codes ??
818   ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
819   Old Hungarian,Old Hungarian,Lydian,Mandaean,Manichaean,%
820   Manichaean,Meroitic Cursive,Meroitic,Old North Arabian,%
821   Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
822   Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
823   Old South Arabian,%
824 \def\bbl@provide@dirs#1{%
825   \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts\bbl@rscripts}%
826   \ifin@
827     \global\bbl@csarg\chardef{wdir@#1}\@ne
828     \bbl@xin@{\csname bbl@sname@#1\endcsname}{\bbl@alscripts}%
829     \ifin@
830       \global\bbl@csarg\chardef{wdir@#1}\tw@ % useless in xetex
831     \fi
832   \else
833     \global\bbl@csarg\chardef{wdir@#1}\z@
834   \fi
835   \ifodd\bbl@engine
836     \bbl@csarg\ifcase{wdir@#1}%
837       \directlua{ Babel.locale_props[\the\localeid].textdir = 'l' }%
838     \or
839       \directlua{ Babel.locale_props[\the\localeid].textdir = 'r' }%
840     \or
841       \directlua{ Babel.locale_props[\the\localeid].textdir = 'al' }%
842     \fi
843   \fi}

```

```

844 \def\bbl@switchdir{%
845   \bbl@ifunset{bbl@lsys@\languagename}{\bbl@provide@lsys{\languagename}}{}}%
846   \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}}%
847   \bbl@exp{\bbl@setdirs\bbl@cl{wdir}}}%
848 \def\bbl@setdirs#1{% TODO - math
849   \ifcase\bbl@select@type % TODO - strictly, not the right test
850     \bbl@bodydir{#1}%
851     \bbl@pdir{#1}%
852   \fi
853   \bbl@texdir{#1}}
854 % TODO. Only if \bbl@bidimode > 0?:
855 \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
856 \DisableBabelHook{babel-bidi}

```

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

```

857 \ifodd\bbl@engine % luatex=1
858   \chardef\bbl@thetextdir\z@
859   \chardef\bbl@thepardir\z@
860   \def\bbl@getluadir#1{%
861     \directlua{
862       if tex.#1dir == 'TLT' then
863         tex.sprint('0')
864       elseif tex.#1dir == 'TRT' then
865         tex.sprint('1')
866       end}}
867   \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\texdir.. 3=0 lr/1 r1
868     \ifcase#3\relax
869       \ifcase\bbl@getluadir{#1}\relax\else
870         #2 TLT\relax
871       \fi
872     \else
873       \ifcase\bbl@getluadir{#1}\relax
874         #2 TRT\relax
875       \fi
876     \fi}
877   \def\bbl@texdir#1{%
878     \bbl@setluadir{tex}\texdir{#1}%
879     \chardef\bbl@thetextdir#1\relax
880     \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*3+#1}}
881   \def\bbl@pdir#1{%
882     \bbl@setluadir{par}\pardir{#1}%
883     \chardef\bbl@thepardir#1\relax}
884   \def\bbl@bodydir{\bbl@setluadir{body}\bodydir}
885   \def\bbl@pagedir{\bbl@setluadir{page}\pagedir}
886   \def\bbl@dirparastext{\pardir\the\texdir\relax}% %%%
887   % Sadly, we have to deal with boxes in math with basic.
888   % Activated every math with the package option bidi=:
889   \def\bbl@mathboxdir{%
890     \ifcase\bbl@thetextdir\relax
891       \everyhbox{\texdir TLT\relax}%
892     \else
893       \everyhbox{\texdir TRT\relax}%
894     \fi}
895   \frozen@everymath\expandafter{%
896     \expandafter\bbl@mathboxdir\the\frozen@everymath}
897   \frozen@everydisplay\expandafter{%
898     \expandafter\bbl@mathboxdir\the\frozen@everydisplay}
899 \else % pdftex=0, xetex=2
900   \newcount\bbl@dirlevel

```

```

901 \chardef\bbl@thetextdir\z@
902 \chardef\bbl@thepardir\z@
903 \def\bbl@textdir#1{%
904   \ifcase#1\relax
905     \chardef\bbl@thetextdir\z@
906     \bbl@textdir@i\beginL\endL
907   \else
908     \chardef\bbl@thetextdir\@ne
909     \bbl@textdir@i\beginR\endR
910   \fi}
911 \def\bbl@textdir@i#1#2{%
912   \ifhmode
913     \ifnum\currentgrouplevel>\z@
914       \ifnum\currentgrouplevel=\bbl@dirlevel
915         \bbl@error{Multiple bidi settings inside a group}%
916         {I'll insert a new group, but expect wrong results.}%
917         \bgroup\aftergroup#2\aftergroup\egroup
918       \else
919         \ifcase\currentgrouptype\or % 0 bottom
920           \aftergroup#2% 1 simple {}
921         \or
922           \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
923         \or
924           \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
925         \or\or\or % vbox vtop align
926         \or
927           \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
928         \or\or\or\or\or\or % output math disc insert vcent mathchoice
929         \or
930           \aftergroup#2% 14 \begingroup
931         \else
932           \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
933         \fi
934       \fi
935       \bbl@dirlevel\currentgrouplevel
936     \fi
937     #1%
938   \fi}
939 \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
940 \let\bbl@bodydir\@gobble
941 \let\bbl@pagedir\@gobble
942 \def\bbl@dirparastext{\chardef\bbl@thepardir\bbl@thetextdir}

```

The following command is executed only if there is a right-to-left script (once). It activates the `\everypar` hack for xetex, to properly handle the par direction. Note text and par dirs are decoupled to some extent (although not completely).

```

943 \def\bbl@xebidipar{%
944   \let\bbl@xebidipar\relax
945   \TeXeTstate\@ne
946   \def\bbl@xeverypar{%
947     \ifcase\bbl@thepardir
948       \ifcase\bbl@thetextdir\else\beginR\fi
949     \else
950       {\setbox\z@\lastbox\beginR\box\z@}%
951     \fi}%
952   \let\bbl@severypar\everypar
953   \newtoks\everypar
954   \everypar=\bbl@severypar
955   \bbl@severypar{\bbl@xeverypar\the\everypar}}

```

```

956 \ifnum\bbbl@bidimode>200
957 \let\bbbl@textdir@i\@gobbletwo
958 \let\bbbl@xebidipar\@empty
959 \AddBabelHook{bidi}{foreign}{%
960 \def\bbbl@tempa{\def\BabelText####1}%
961 \ifcase\bbbl@thetextdir
962 \expandafter\bbbl@tempa\expandafter{\BabelText{\LR{##1}}}%
963 \else
964 \expandafter\bbbl@tempa\expandafter{\BabelText{\RL{##1}}}%
965 \fi}
966 \def\bbbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}
967 \fi
968 \fi

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

969 \DeclareRobustCommand\babelsublr[1]{\leavevmode{\bbbl@textdir\z@#1}}
970 \AtBeginDocument{%
971 \ifx\pdfstringdefDisableCommands\@undefined\else
972 \ifx\pdfstringdefDisableCommands\relax\else
973 \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
974 \fi
975 \fi}

```

## 7.10 Local Language Configuration

`\loadlocalcfg` At some sites it may be necessary to add site-specific actions to a language definition file. This can be done by creating a file with the same name as the language definition file, but with the extension `.cfg`. For instance the file `norsk.cfg` will be loaded when the language definition file `norsk.ldf` is loaded.

For plain-based formats we don't want to override the definition of `\loadlocalcfg` from `plain.def`.

```

976 \bbbl@trace{Local Language Configuration}
977 \ifx\loadlocalcfg\@undefined
978 \@ifpackagewith{babel}{noconfigs}%
979 {\let\loadlocalcfg\@gobble}%
980 {\def\loadlocalcfg#1{%
981 \InputIfFileExists{#1.cfg}%
982 {\typeout{*****^J%
983 * Local config file #1.cfg used^^J%
984 *}}}%
985 \@empty}}
986 \fi

```

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

```

987 \ifx\@unexpandable@protect\@undefined
988 \def\@unexpandable@protect{\noexpand\protect\noexpand}
989 \long\def\protected@write#1#2#3{%
990 \begingroup
991 \let\thepage\relax
992 #2%
993 \let\protect\@unexpandable@protect
994 \edef\reserved@a{\write#1{#3}}%
995 \reserved@a
996 \endgroup
997 \if@nobreak\ifvmode\nobreak\fi\fi}
998 \fi
999 %
1000 % \subsection{Language options}

```

```

1001%
1002% Languages are loaded when processing the corresponding option
1003% \textit{except} if a |main| language has been set. In such a
1004% case, it is not loaded until all options has been processed.
1005% The following macro inputs the ldf file and does some additional
1006% checks (|\input| works, too, but possible errors are not caught).
1007%
1008% \begin{macrocode}
1009 \bbl@trace{Language options}
1010 \let\bbl@afterlang\relax
1011 \let\BabelModifiers\relax
1012 \let\bbl@loaded@empty
1013 \def\bbl@load@language#1{%
1014 \InputIfFileExists{#1.ldf}%
1015 {\edef\bbl@loaded{\CurrentOption
1016 \ifx\bbl@loaded@empty\else,\bbl@loaded\fi}%
1017 \expandafter\let\expandafter\bbl@afterlang
1018 \csname\CurrentOption.ldf-h@@k\endcsname
1019 \expandafter\let\expandafter\BabelModifiers
1020 \csname bbl@mod@\CurrentOption\endcsname}%
1021 {\bbl@error{%
1022 Unknown option '\CurrentOption'. Either you misspelled it\\%
1023 or the language definition file \CurrentOption.ldf was not found}}{%
1024 Valid options are, among others: shorthands=, KeepShorthandsActive,\\%
1025 activeacute, activegrave, noconfigs, safe=, main=, math=\\%
1026 headfoot=, strings=, config=, hyphenmap=, or a language name.}}}

```

Now, we set a few language options whose names are different from ldf files. These declarations are preserved for backwards compatibility, but they must be eventually removed. Use proxy files instead.

```

1027 \def\bbl@try@load@lang#1#2#3{%
1028 \IfFileExists{\CurrentOption.ldf}%
1029 {\bbl@load@language{\CurrentOption}}}%
1030 {#1\bbl@load@language{#2}#3}}
1031 \DeclareOption{hebrew}{%
1032 \input{rlbabel.def}%
1033 \bbl@load@language{hebrew}}
1034 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}{}}
1035 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}{}}
1036 \DeclareOption{ nynorsk}{\bbl@try@load@lang{}{norsk}{}}
1037 \DeclareOption{polutonikogreek}{%
1038 \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}
1039 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}{}}
1040 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}{}}
1041 \DeclareOption{uppersorbian}{\bbl@try@load@lang{}{usorbian}{}}

```

Another way to extend the list of ‘known’ options for babel was to create the file `bblopts.cfg` in which one can add option declarations. However, this mechanism is deprecated – if you want an alternative name for a language, just create a new `.ldf` file loading the actual one. You can also set the name of the file with the package option `config=<name>`, which will load `<name>.cfg` instead.

```

1042 \ifx\bbl@opt@config@nnil
1043 \ifpackagewith{babel}{noconfigs}{}%
1044 {\InputIfFileExists{bblopts.cfg}%
1045 {\typeout{*****^J%
1046 * Local config file bblopts.cfg used^^J%
1047 *}}}%
1048 {}}%
1049 \else
1050 \InputIfFileExists{\bbl@opt@config.cfg}%

```



```

1051 {\typeout{*****^J%
1052          * Local config file \bbl@opt@config.cfg used^^J%
1053          *}}%
1054 {\bbl@error{%
1055     Local config file '\bbl@opt@config.cfg' not found}{%
1056     Perhaps you misspelled it.}}%
1057 \fi

```

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

```

1058 \let\bbl@tempc\relax
1059 \bbl@foreach\bbl@language@opts{%
1060   \ifcase\bbl@iniflag % Default
1061     \bbl@ifunset{ds@#1}%
1062     {\DeclareOption{#1}{\bbl@load@language{#1}}}%
1063     {}%
1064   \or % provide=*
1065     \@gobble % case 2 same as 1
1066   \or % provide+=*
1067     \bbl@ifunset{ds@#1}%
1068     {\IfFileExists{#1.ldf}}{%
1069       {\IfFileExists{babel-#1.tex}}{\@namedef{ds@#1}}}%
1070     {}%
1071     \bbl@ifunset{ds@#1}%
1072     {\def\bbl@tempc{#1}%
1073       \DeclareOption{#1}{%
1074         \ifnum\bbl@iniflag>\@ne
1075           \bbl@ldfinit
1076           \babelprovide[import]{#1}%
1077           \bbl@afterldf}%
1078         \else
1079           \bbl@load@language{#1}%
1080         \fi}}%
1081     {}%
1082   \or % provide*=*
1083     \def\bbl@tempc{#1}%
1084     \bbl@ifunset{ds@#1}%
1085     {\DeclareOption{#1}{%
1086       \bbl@ldfinit
1087       \babelprovide[import]{#1}%
1088       \bbl@afterldf}}}%
1089     {}%
1090   \fi}

```

Now, we make sure an option is explicitly declared for any language set as global option, by checking if an `ldf` exists. The previous step was, in fact, somewhat redundant, but that way we minimize accessing the file system just to see if the option could be a language.

```

1091 \let\bbl@tempb\@nnil
1092 \bbl@foreach\@classoptionslist{%
1093   \bbl@ifunset{ds@#1}%
1094   {\IfFileExists{#1.ldf}}{%
1095     {\IfFileExists{babel-#1.tex}}{\@namedef{ds@#1}}}%
1096   {}%
1097   \bbl@ifunset{ds@#1}%
1098   {\def\bbl@tempb{#1}%
1099     \DeclareOption{#1}{%
1100       \ifnum\bbl@iniflag>\@ne

```

```

1101      \bbl@ldfinit
1102      \babelprovide[import]{#1}%
1103      \bbl@afterldf{}%
1104      \else
1105        \bbl@load@language{#1}%
1106      \fi}%
1107    {}

```

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

```

1108 \ifnum\bbl@iniflag=\z@ \else
1109   \ifx\bbl@opt@main\@nnil
1110     \ifx\bbl@tempc\relax
1111       \let\bbl@opt@main\bbl@tempb
1112     \else
1113       \let\bbl@opt@main\bbl@tempc
1114     \fi
1115   \fi
1116 \fi
1117 \ifx\bbl@opt@main\@nnil \else
1118   \expandafter
1119   \let\expandafter\bbl@loadmain\csname ds@\bbl@opt@main\endcsname
1120   \expandafter\let\csname ds@\bbl@opt@main\endcsname\@empty
1121 \fi

```

And we are done, because all options for this pass has been declared. Those already processed in the first pass are just ignored.

The options have to be processed in the order in which the user specified them (except, of course, global options, which  $\TeX$  processes before):

```

1122 \def\AfterBabelLanguage#1{%
1123   \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang}{}}
1124 \DeclareOption*{}
1125 \ProcessOptions*

```

This finished the second pass. Now the third one begins, which loads the main language set with the key main. A warning is raised if the main language is not the same as the last named one, or if the value of the key main is not a language. Then execute directly the option (because it could be used only in main). After loading all languages, we deactivate \AfterBabelLanguage.

```

1126 \bbl@trace{Option 'main'}
1127 \ifx\bbl@opt@main\@nnil
1128   \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}
1129   \let\bbl@tempc\@empty
1130   \bbl@for\bbl@tempb\bbl@tempa{%
1131     \bbl@xin@{,\bbl@tempb,}{,\bbl@loaded,}%
1132     \ifin@\edef\bbl@tempc{\bbl@tempb}\fi}
1133   \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
1134   \expandafter\bbl@tempa\bbl@loaded,\@nnil
1135   \ifx\bbl@tempb\bbl@tempc \else
1136     \bbl@warning{%
1137       Last declared language option is ``\bbl@tempc'',\%
1138       but the last processed one was ``\bbl@tempb'.\%
1139       The main language cannot be set as both a global\%
1140       and a package option. Use `main=\bbl@tempc' as\%
1141       option. Reported}%
1142   \fi
1143 \else
1144   \ifodd\bbl@iniflag % case 1,3
1145     \bbl@ldfinit
1146     \let\CurrentOption\bbl@opt@main
1147     \bbl@exp{\babelprovide[import,main]{\bbl@opt@main}}

```

```

1148 \bbl@afterldf{%
1149 \else % case 0,2
1150 \chardef\bbl@iniflag\z@ % Force ldf
1151 \expandafter\let\csname ds@\bbl@opt@main\endcsname\bbl@loadmain
1152 \ExecuteOptions{\bbl@opt@main}
1153 \DeclareOption*{%
1154 \ProcessOptions*
1155 \fi
1156 \fi
1157 \def\AfterBabelLanguage{%
1158 \bbl@error
1159 {Too late for \string\AfterBabelLanguage}%
1160 {Languages have been loaded, so I can do nothing}}

```

In order to catch the case where the user forgot to specify a language we check whether `\bbl@main@language`, has become defined. If not, no language has been loaded and an error message is displayed.

```

1161 \ifx\bbl@main@language\@undefined
1162 \bbl@info{%
1163 You haven't specified a language. I'll use 'nil'\%
1164 as the main language. Reported}
1165 \bbl@load@language{nil}
1166 \fi
1167 \</package>
1168 \<core>

```

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

The kernel of the babel system is currently stored in `babel.def`. The file `babel.def` contains most of the code. The file `hyphen.cfg` is a file that can be loaded into the format, which is necessary when you want to be able to switch hyphenation patterns.

Because plain  $\TeX$  users might want to use some of the features of the babel system too, care has to be taken that plain  $\TeX$  can process the files. For this reason the current format will have to be checked in a number of places. Some of the code below is common to plain  $\TeX$  and  $\LaTeX$ , some of it is for the  $\LaTeX$  case only.

Plain formats based on `etex` (`etex`, `xetex`, `luatex`) don't load `hyphen.cfg` but `etex.src`, which follows a different naming convention, so we need to define the babel names. It presumes `language.def` exists and it is the same file used when formats were created.

### 8.1 Tools

```

1169 \ifx\ldf@quit\@undefined\else
1170 \endinput\fi % Same line!
1171 \<<Make sure ProvidesFile is defined>>
1172 \ProvidesFile{babel.def}[\<<date>> \<<version>> Babel common definitions]

```

The file `babel.def` expects some definitions made in the  $\LaTeX 2_{\epsilon}$  style file. So, In  $\LaTeX 2.09$  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 an alternative mechanism is provided. For the moment, only `\babeloptionstrings` and `\babeloptionmath` are provided, which can be defined before loading babel. `\BabelModifiers` can be set too (but not sure it works).

```

1173 \ifx\AtBeginDocument\@undefined % TODO. change test.
1174 \<<Emulate LaTeX>>
1175 \def\languagename{english}%
1176 \let\bbl@opt@shorthands\@nnil
1177 \def\bbl@ifshorthand#1#2#3{#2}%
1178 \let\bbl@language@opts\@empty
1179 \ifx\babeloptionstrings\@undefined
1180 \let\bbl@opt@strings\@nnil

```

```

1181 \else
1182   \let\bbl@opt@strings\babeloptionstrings
1183 \fi
1184 \def\BabelStringsDefault{generic}
1185 \def\bbl@tempa{normal}
1186 \ifx\babeloptionmath\bbl@tempa
1187   \def\bbl@mathnormal{\noexpand\textormath}
1188 \fi
1189 \def\AfterBabelLanguage#1#2{}
1190 \ifx\BabelModifiers\@undefined\let\BabelModifiers\relax\fi
1191 \let\bbl@afterlang\relax
1192 \def\bbl@opt@safe{BR}
1193 \ifx\@uclclist\@undefined\let\@uclclist\@empty\fi
1194 \ifx\bbl@trace\@undefined\def\bbl@trace#1{}\fi
1195 \expandafter\newif\csname ifbbl@single\endcsname
1196 \chardef\bbl@bidimode\z@
1197 \fi

```

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

```

1198 \ifx\bbl@trace\@undefined
1199   \let\LdfInit\endinput
1200   \def\ProvidesLanguage#1{\endinput}
1201 \endinput\fi % Same line!

```

And continue.

## 9 Multiple languages

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

Plain T<sub>E</sub>X version 3.0 provides the primitive `\language` that is used to store the current language.

When used with a pre-3.0 version this function has to be implemented by allocating a counter.

```

1202 <<Define core switching macros>>

```

`\adddialect` The macro `\adddialect` can be used to add the name of a dialect or variant language, for which an already defined hyphenation table can be used.

```

1203 \def\bbl@version{<<version>>)}
1204 \def\bbl@date{<<date>>)}
1205 \def\adddialect#1#2{%
1206   \global\chardef#1#2\relax
1207   \bbl@usehooks{adddialect}{#1}{#2}}%
1208   \begingroup
1209     \count@#1\relax
1210     \def\bbl@elt##1##2##3##4{%
1211       \ifnum\count@=##2\relax
1212         \bbl@info{\string#1 = using hyphenrules for ##1\\%
1213           (\string\language\the\count@)}%
1214         \def\bbl@elt####1####2####3####4}%
1215       \fi}%
1216   \bbl@cs{languages}%
1217   \endgroup}

```

`\bbl@iflanguage` executes code only if the language `l@` exists. Otherwise raises an 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 intended 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.

```

1218 \def\bbl@fixname#1{%
1219   \begingroup

```

```

1220 \def\bbl@tempe{#1}%
1221 \edef\bbl@tempd{\noexpand\@ifundefined{\noexpand\bbl@tempe{#1}}}%
1222 \bbl@tempd
1223 {\lowercase\expandafter{\bbl@tempd}%
1224 {\uppercase\expandafter{\bbl@tempd}%
1225 \@empty
1226 {\edef\bbl@tempd{\def\noexpand#1{#1}}}%
1227 {\uppercase\expandafter{\bbl@tempd}}}%
1228 {\edef\bbl@tempd{\def\noexpand#1{#1}}}%
1229 {\lowercase\expandafter{\bbl@tempd}}}%
1230 \@empty
1231 \edef\bbl@tempd{\endgroup\def\noexpand#1{#1}}%
1232 \bbl@tempd
1233 \bbl@exp{\bbl@usehooks{language}{\language}{#1}}%
1234 \def\bbl@iflanguage#1{%
1235 \@ifundefined{#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@bcpllookup` either returns the found ini or it is `\relax`.

```

1236 \def\bbl@bcpcase#1#2#3#4\@#5{%
1237 \ifx\@empty#3%
1238 \uppercase{\def#5{#1#2}}%
1239 \else
1240 \uppercase{\def#5{#1}}%
1241 \lowercase{\edef#5{#5#2#3#4}}%
1242 \fi}
1243 \def\bbl@bcpllookup#1-#2-#3-#4\@#5{%
1244 \let\bbl@bcp\relax
1245 \lowercase{\def\bbl@tempa{#1}}%
1246 \ifx\@empty#2%
1247 \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
1248 \else\ifx\@empty#3%
1249 \bbl@bcpcase#2\@empty\@empty\@#5\bbl@tempb
1250 \IfFileExists{babel-\bbl@tempa-\bbl@tempb.ini}%
1251 {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb}}%
1252 {}%
1253 \ifx\bbl@bcp\relax
1254 \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
1255 \fi
1256 \else
1257 \bbl@bcpcase#2\@empty\@empty\@#5\bbl@tempb
1258 \bbl@bcpcase#3\@empty\@empty\@#5\bbl@tempc
1259 \IfFileExists{babel-\bbl@tempa-\bbl@tempb-\bbl@tempc.ini}%
1260 {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb-\bbl@tempc}}%
1261 {}%
1262 \ifx\bbl@bcp\relax
1263 \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
1264 {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
1265 {}%
1266 \fi
1267 \ifx\bbl@bcp\relax
1268 \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
1269 {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
1270 {}%
1271 \fi
1272 \ifx\bbl@bcp\relax

```

```

1273 \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
1274 \fi
1275 \fi\fi}
1276 \let\bbl@initoload\relax
1277 \def\bbl@provide@locale{%
1278 \ifx\babelprovide\undefined
1279 \bbl@error{For a language to be defined on the fly 'base'\\%
1280 is not enough, and the whole package must be\\%
1281 loaded. Either delete the 'base' option or\\%
1282 request the languages explicitly}%
1283 {See the manual for further details.}%
1284 \fi
1285 % TODO. Option to search if loaded, with \LocaleForEach
1286 \let\bbl@auxname\language % Still necessary. TODO
1287 \bbl@ifunset{bbl@bcp@map@\language}{}% Move uplevel??
1288 {\edef\language{\@nameuse{bbl@bcp@map@\language}}}%
1289 \ifbbl@bcpallowed
1290 \expandafter\ifx\csname date\language\endcsname\relax
1291 \expandafter
1292 \bbl@bcplookup\language-\@empty-\@empty-\@empty\@
1293 \ifx\bbl@bcp\relax\else % Returned by \bbl@bcplookup
1294 \edef\language{\bbl@bcp@prefix\bbl@bcp}%
1295 \edef\localename{\bbl@bcp@prefix\bbl@bcp}%
1296 \expandafter\ifx\csname date\language\endcsname\relax
1297 \let\bbl@initoload\bbl@bcp
1298 \bbl@exp{\bbl@babelprovide[\bbl@autoload@bcptoptions]{\language}}%
1299 \let\bbl@initoload\relax
1300 \fi
1301 \bbl@csarg\xdef{bcp@map@\bbl@bcp}{\localename}%
1302 \fi
1303 \fi
1304 \fi
1305 \expandafter\ifx\csname date\language\endcsname\relax
1306 \IfFileExists{babel-\language.tex}%
1307 {\bbl@exp{\bbl@babelprovide[\bbl@autoload@options]{\language}}}%
1308 {}}%
1309 \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.

```

1310 \def\iflanguage#1{%
1311 \bbl@iflanguage{#1}{%
1312 \ifnum\csname l@#1\endcsname=\language
1313 \expandafter\@firstoftwo
1314 \else
1315 \expandafter\@secondoftwo
1316 \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.

```

1317 \let\bbl@select@type\z@
1318 \edef\selectlanguage{%
1319 \noexpand\protect
1320 \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`.

```
1321 \ifx\@undefined\protect\let\protect\relax\fi
```

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

```
1322 \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.

```
1323 \def\bbl@language@stack{}
```

When using a stack we need a mechanism to push an element on the stack and to retrieve the information afterwards.

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

```
1324 \def\bbl@push@language{%
1325   \ifx\language\@undefined\else
1326     \xdef\bbl@language@stack{\language+\bbl@language@stack}%
1327   \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 `\language`. For this we first define a helper function.

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

```
1328 \def\bbl@pop@lang#1+#2\@@{%
1329   \edef\language{#1}%
1330   \xdef\bbl@language@stack{#2}}
```

The reason for the somewhat weird arrangement of arguments to the helper function is the fact it is called in the following way. This means that before `\bbl@pop@lang` is executed TeX first *expands* the stack, stored in `\bbl@language@stack`. The result of that is that the argument string of `\bbl@pop@lang` contains one or more language names, each followed by a '+'-sign (zero language names won't occur as this macro will only be called after something has been pushed on the stack).

```
1331 \let\bbl@ifrestoring\@secondoftwo
1332 \def\bbl@pop@language{%
1333   \expandafter\bbl@pop@lang\bbl@language@stack\@@
1334   \let\bbl@ifrestoring\@firstoftwo
1335   \expandafter\bbl@set@language\expandafter{\language}%
1336   \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).

```

1337 \chardef\localeid\z@
1338 \def\bbl@id@last{0} % No real need for a new counter
1339 \def\bbl@id@assign{%
1340   \bbl@ifunset{bbl@id@@\language}%
1341   {\count@bbl@id@last\relax
1342    \advance\count@one
1343    \bbl@csarg\chardef{id@@\language}\count@
1344    \edef\bbl@id@last{\the\count@}%
1345    \ifcase\bbl@engine\or
1346      \directlua{
1347        Babel = Babel or {}
1348        Babel.locale_props = Babel.locale_props or {}
1349        Babel.locale_props[\bbl@id@last] = {}
1350        Babel.locale_props[\bbl@id@last].name = '\language'
1351      }%
1352    \fi}%
1353  }%
1354  \chardef\localeid\bbl@c1{id@}}

```

The unprotected part of \selectlanguage.

```

1355 \expandafter\def\csname selectlanguage \endcsname#1{%
1356   \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\tw@\fi
1357   \bbl@push@language
1358   \aftergroup\bbl@pop@language
1359   \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 historical 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 `\language` 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.

```

1360 \def\BabelContentsFiles{toc,lof,lot}
1361 \def\bbl@set@language#1{% from selectlanguage, pop@
1362   % The old buggy way. Preserved for compatibility.
1363   \edef\language{%
1364     \ifnum\escapechar=\expandafter`\string#1\@empty
1365     \else\string#1\@empty\fi}%
1366   \ifcat\relax\noexpand#1%
1367     \expandafter\ifx\csname date\language\endcsname\relax
1368       \edef\language{#1}%
1369       \let\localename\language
1370     \else
1371       \bbl@info{Using '\string\language' instead of 'language' is\\%
1372         deprecated. If what you want is to use a\\%
1373         macro containing the actual locale, make\\%
1374         sure it does not not match any language.\\%
1375         Reported}%
1376       I'll\\%
1377       try to fix '\string\localename', but I cannot promise\\%
1378       anything. Reported}%
1379     \ifx\scantokens\undefined
1380       \def\localename{??}%
1381     \else
1382       \scantokens\expandafter{\expandafter
1383         \def\expandafter\localename\expandafter{\language}}%
1384     \fi

```



```

1385 \fi
1386 \else
1387 \def\localename{#1}% This one has the correct catcodes
1388 \fi
1389 \select@language{\languagename}%
1390 % write to auxs
1391 \expandafter\ifx\csname date\languagename\endcsname\relax\else
1392 \if@filesw
1393 \ifx\babel@aux\@gobbletwo\else % Set if single in the first, redundant
1394 % \bbl@savelastskip
1395 \protected@write\@auxout{}\string\babel@aux{\bbl@auxname}{}}%
1396 % \bbl@restorelastskip
1397 \fi
1398 \bbl@usehooks{write}{}}%
1399 \fi
1400 \fi}
1401 % The following is used above to deal with skips before the write
1402 % whatsit. Adapted from hyperref, but it might fail, so for the moment
1403 % it's not activated. TODO.
1404 \def\bbl@savelastskip{%
1405 \let\bbl@restorelastskip\relax
1406 \ifvmode
1407 \ifdim\lastskip=\z@
1408 \let\bbl@restorelastskip\nobreak
1409 \else
1410 \bbl@exp{%
1411 \def\\bbl@restorelastskip{%
1412 \skip@=\the\lastskip
1413 \\nobreak \vskip-\skip@ \vskip\skip@}}%
1414 \fi
1415 \fi}
1416 \newif\ifbbl@bcpallowed
1417 \bbl@bcpallowedfalse
1418 \def\select@language#1{% from set@, babel@aux
1419 % set hymap
1420 \ifnum\bbl@hymapsel=\@ccclv\chardef\bbl@hymapsel4\relax\fi
1421 % set name
1422 \edef\languagename{#1}%
1423 \bbl@fixname\languagename
1424 % TODO. name@map must be here?
1425 \bbl@provide@locale
1426 \bbl@iflanguage\languagename{%
1427 \expandafter\ifx\csname date\languagename\endcsname\relax
1428 \bbl@error
1429 {Unknown language '\languagename'. Either you have\\%
1430 misspelled its name, it has not been installed,\\%
1431 or you requested it in a previous run. Fix its name,\\%
1432 install it or just rerun the file, respectively. In\\%
1433 some cases, you may need to remove the aux file}%
1434 {You may proceed, but expect wrong results}}%
1435 \else
1436 % set type
1437 \let\bbl@select@type\z@
1438 \expandafter\bbl@switch\expandafter{\languagename}%
1439 \fi}}
1440 \def\babel@aux#1#2{% TODO. See how to avoid undefined nil's
1441 \select@language{#1}%
1442 \bbl@foreach\BabelContentsFiles{%
1443 \@writefile{##1}{\babel@toc{#1}{#2}}}% % TODO - ok in plain?

```

```

1444 \def\babel@toc#1#2{%
1445   \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 `\language`.

Then we have to redefine `\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.

```

1446 \newif\ifbbl@usedatagroup
1447 \def\bbl@switch#1{% from select@, foreign@
1448   % make sure there is info for the language if so requested
1449   \bbl@ensureinfo{#1}%
1450   % restore
1451   \originalTeX
1452   \expandafter\def\expandafter\originalTeX\expandafter{%
1453     \csname noextras#1\endcsname
1454     \let\originalTeX\empty
1455     \babel@beginsave}%
1456   \bbl@usehooks{afterreset}{}%
1457   \languageshorthands{none}%
1458   % set the locale id
1459   \bbl@id@assign
1460   % switch captions, date
1461   % No text is supposed to be added here, so we remove any
1462   % spurious spaces.
1463   \bbl@bsphack
1464   \ifcase\bbl@select@type
1465     \csname captions#1\endcsname\relax
1466     \csname date#1\endcsname\relax
1467   \else
1468     \bbl@xin@{,captions,}{, \bbl@select@opts,}%
1469     \ifin@
1470       \csname captions#1\endcsname\relax
1471     \fi
1472     \bbl@xin@{,date,}{, \bbl@select@opts,}%
1473     \ifin@ % if \foreign... within \<lang>date
1474       \csname date#1\endcsname\relax
1475     \fi
1476   \fi
1477   \bbl@esphack
1478   % switch extras
1479   \bbl@usehooks{beforeextras}{}%
1480   \csname extras#1\endcsname\relax
1481   \bbl@usehooks{afterextras}{}%
1482   % > babel-ensure
1483   % > babel-sh-<short>
1484   % > babel-bidi
1485   % > babel-fontspec
1486   % hyphenation - case mapping
1487   \ifcase\bbl@opt@hyphenmap\or
1488     \def\BabelLower##1##2{\lccode##1=##2\relax}%
1489     \ifnum\bbl@hymapsel>4\else

```

```

1490 \csname\language @bbl@hyphenmap\endcsname
1491 \fi
1492 \chardef\bbl@opt@hyphenmap\z@
1493 \else
1494 \ifnum\bbl@hymapsel>\bbl@opt@hyphenmap\else
1495 \csname\language @bbl@hyphenmap\endcsname
1496 \fi
1497 \fi
1498 \let\bbl@hymapsel\@cclv
1499 % hyphenation - select patterns
1500 \bbl@patterns{#1}%
1501 % hyphenation - allow stretching with babelnohyphens
1502 \ifnum\language=\l@babelnohyphens
1503 \babel@savevariable\emergencystretch
1504 \emergencystretch\maxdimen
1505 \babel@savevariable\hbadness
1506 \hbadness\@M
1507 \fi
1508 % hyphenation - mins
1509 \babel@savevariable\lefthyphenmin
1510 \babel@savevariable\righthyphenmin
1511 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
1512 \set@hyphenmins\tw@\thr@\relax
1513 \else
1514 \expandafter\expandafter\expandafter\set@hyphenmins
1515 \csname #1hyphenmins\endcsname\relax
1516 \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.

```

1517 \long\def\otherlanguage#1{%
1518 \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\thr@\fi
1519 \csname selectlanguage \endcsname{#1}%
1520 \ignorespaces}

```

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

```

1521 \long\def\endotherlanguage{%
1522 \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`.

```

1523 \expandafter\def\csname otherlanguage*\endcsname{%
1524 \@ifnextchar[\bbl@otherlanguage@s{\bbl@otherlanguage@s[]}}
1525 \def\bbl@otherlanguage@s[#1]#2{%
1526 \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
1527 \def\bbl@select@opts{#1}%
1528 \foreign@language{#2}}

```

At the end of the environment we need to switch off the extra definitions. The grouping mechanism of the environment will take care of resetting the correct hyphenation rules and “extras”.

```

1529 \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.

```

1530 \providecommand\bbl@beforeforeign{}
1531 \edef\foreignlanguage{%
1532   \noexpand\protect
1533   \expandafter\noexpand\csname foreignlanguage \endcsname}
1534 \expandafter\def\csname foreignlanguage \endcsname{%
1535   \@ifstar\bbl@foreign@s\bbl@foreign@x}
1536 \providecommand\bbl@foreign@x[3][]{%
1537   \begingroup
1538     \def\bbl@select@opts{#1}%
1539     \let\BabelText\@firstofone
1540     \bbl@beforeforeign
1541     \foreign@language{#2}%
1542     \bbl@usehooks{foreign}{}%
1543     \BabelText{#3}% Now in horizontal mode!
1544   \endgroup}
1545 \def\bbl@foreign@s#1#2{% TODO - \shapemode, \setpar, ?\@par
1546   \begingroup
1547     {\par}%
1548     \let\bbl@select@opts\@empty
1549     \let\BabelText\@firstofone
1550     \foreign@language{#1}%
1551     \bbl@usehooks{foreign*}{}%
1552     \bbl@dirparastext
1553     \BabelText{#2}% Still in vertical mode!
1554     {\par}%
1555   \endgroup}

```

`\foreign@language` This macro does the work for `\foreignlanguage` and the other `language*` 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`.

```

1556 \def\foreign@language#1{%
1557   % set name
1558   \edef\language#1}%
1559 \ifbbl@usedategroup
1560   \bbl@add\bbl@select@opts{,date,}%
1561   \bbl@usedategroupfalse
1562 \fi

```

```

1563 \bbl@fixname\language\language
1564 % TODO. name@map here?
1565 \bbl@provide@locale
1566 \bbl@iflanguage\language\language{%
1567   \expandafter\ifx\csname date\language\endcsname\relax
1568     \bbl@warning % TODO - why a warning, not an error?
1569     {Unknown language `#1'. Either you have\\%
1570      misspelled its name, it has not been installed,\\%
1571      or you requested it in a previous run. Fix its name,\\%
1572      install it or just rerun the file, respectively. In\\%
1573      some cases, you may need to remove the aux file.\\%
1574      I'll proceed, but expect wrong results.\\%
1575      Reported}%
1576   \fi
1577   % set type
1578   \let\bbl@select@type\@ne
1579   \expandafter\bbl@switch\expandafter{\language}}

```

**\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.

```

1580 \let\bbl@hyphlist\@empty
1581 \let\bbl@hyphenation@\relax
1582 \let\bbl@pttnlist\@empty
1583 \let\bbl@patterns@\relax
1584 \let\bbl@hymapsel=\@cclv
1585 \def\bbl@patterns#1{%
1586   \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
1587     \csname l@#1\endcsname
1588     \edef\bbl@tempa{#1}%
1589   \else
1590     \csname l@#1:\f@encoding\endcsname
1591     \edef\bbl@tempa{#1:\f@encoding}%
1592   \fi
1593   \@expandtwoargs\bbl@usehooks{patterns}{#1}{\bbl@tempa}%
1594   % > luatex
1595   \@ifundefined{bbl@hyphenation@}{% Can be \relax!
1596     \begingroup
1597       \bbl@xin@{\number\language,}{\bbl@hyphlist}%
1598     \ifin@else
1599       \@expandtwoargs\bbl@usehooks{hyphenation}{#1}{\bbl@tempa}%
1600       \hyphenation{%
1601         \bbl@hyphenation@
1602         \@ifundefined{bbl@hyphenation@#1}%
1603         \@empty
1604         {\space\csname bbl@hyphenation@#1\endcsname}}%
1605       \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
1606     \fi
1607   \endgroup}}

```

**hyphenrules** The environment `hyphenrules` can be used to select *just* the hyphenation rules. This environment does *not* change `\language` 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*`.

```

1608 \def\hyphenrules#1{%
1609   \edef\bbl@tempf{#1}%
1610   \bbl@fixname\bbl@tempf
1611   \bbl@iflanguage\bbl@tempf{%
1612     \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
1613     \ifx\languageshorthands\@undefined\else
1614       \languageshorthands{none}%
1615     \fi
1616     \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax
1617       \set@hyphenmins\tw@\thr@@\relax
1618     \else
1619       \expandafter\expandafter\expandafter\set@hyphenmins
1620       \csname\bbl@tempf hyphenmins\endcsname\relax
1621     \fi}}
1622 \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.

```

1623 \def\providehyphenmins#1#2{%
1624   \expandafter\ifx\csname #1hyphenmins\endcsname\relax
1625     \@namedef{#1hyphenmins}{#2}%
1626   \fi}

```

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

```

1627 \def\set@hyphenmins#1#2{%
1628   \lefthyphenmin#1\relax
1629   \righthyphenmin#2\relax}

```

`\ProvidesLanguage` The identification code for each file is something that was introduced in  $\text{\LaTeX 2}_{\epsilon}$ . 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.

```

1630 \ifx\ProvidesFile\@undefined
1631   \def\ProvidesLanguage#1[#2 #3 #4]{%
1632     \wlog{Language: #1 #4 #3 <#2>}%
1633   }
1634 \else
1635   \def\ProvidesLanguage#1{%
1636     \begingroup
1637     \catcode`\ 10 %
1638     \@makeother\%
1639     \@ifnextchar[%]
1640       {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}
1641   \def\@provideslanguage#1[#2]{%
1642     \wlog{Language: #1 #2}%
1643     \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
1644     \endgroup}
1645 \fi

```

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

```

1646 \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.

```

1647 \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’:

```

1648 \providecommand\setlocale{%
1649   \bbl@error
1650   {Not yet available}%
1651   {Find an armchair, sit down and wait}}
1652 \let\uselocale\setlocale
1653 \let\locale\setlocale
1654 \let\selectlocale\setlocale
1655 \let\localename\setlocale
1656 \let\textlocale\setlocale
1657 \let\textlanguage\setlocale
1658 \let\languagetext\setlocale

```

## 9.2 Errors

`\@nolanerr` The babel package will signal an error when a documents tries to select a language that hasn’t been 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  $\text{\LaTeX 2}_{\epsilon}$ , 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.

```

1659 \edef\bbl@nulllanguage{\string\language=0}
1660 \ifx\PackageError\@undefined % TODO. Move to Plain
1661   \def\bbl@error#1#2{%
1662     \begingroup
1663     \newlinechar=`^^J
1664     \def\{^^J(babel) }%
1665     \errhelp{#2}\errmessage{\{#1}%
1666     \endgroup}
1667   \def\bbl@warning#1{%
1668     \begingroup
1669     \newlinechar=`^^J
1670     \def\{^^J(babel) }%
1671     \message{\{#1}%
1672     \endgroup}
1673   \let\bbl@infowarn\bbl@warning
1674   \def\bbl@info#1{%
1675     \begingroup
1676     \newlinechar=`^^J
1677     \def\{^^J}%
1678     \wlog{#1}%
1679     \endgroup}
1680 \fi
1681 \def\bbl@nocaption{\protect\bbl@nocaption@i}
1682 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
1683   \global\@namedef{#2}{\textbf{?#1?}}%
1684   \@nameuse{#2}%
1685   \edef\bbl@tempa{#1}%
1686   \bbl@sreplace\bbl@tempa{name}{}}%
1687   \bbl@warning{% TODO.
1688     \@backslashchar#1 not set for '\languagename'. Please,\%
1689     define it after the language has been loaded\%
1690     (typically in the preamble) with:\%

```

```

1691 \string\setlocalecaption{\language}\bbl@tempa}{..}\%
1692 Reported}}
1693 \def\bbl@tentative{\protect\bbl@tentative@i}
1694 \def\bbl@tentative@i#1{%
1695 \bbl@warning{%
1696 Some functions for '#1' are tentative.\%
1697 They might not work as expected and their behavior\%
1698 could change in the future.\%
1699 Reported}}
1700 \def\@nolanerr#1{%
1701 \bbl@error
1702 {You haven't defined the language #1\space yet.\%
1703 Perhaps you misspelled it or your installation\%
1704 is not complete}%
1705 {Your command will be ignored, type <return> to proceed}}
1706 \def\@nopatterns#1{%
1707 \bbl@warning
1708 {No hyphenation patterns were preloaded for\%
1709 the language '#1' into the format.\%
1710 Please, configure your TeX system to add them and\%
1711 rebuild the format. Now I will use the patterns\%
1712 preloaded for \bbl@nulllanguage\space instead}}
1713 \let\bbl@usehooks\@gobbletwo
1714 \ifx\bbl@onlyswitch\@empty\endinput\fi
1715 % Here ended switch.def

Here ended switch.def.

1716 \ifx\directlua\@undefined\else
1717 \ifx\bbl@luapatterns\@undefined
1718 \input luababel.def
1719 \fi
1720 \fi
1721 <<Basic macros>>
1722 \bbl@trace{Compatibility with language.def}
1723 \ifx\bbl@languages\@undefined
1724 \ifx\directlua\@undefined
1725 \openin1 = language.def % TODO. Remove hardcoded number
1726 \ifeof1
1727 \closein1
1728 \message{I couldn't find the file language.def}
1729 \else
1730 \closein1
1731 \begingroup
1732 \def\addlanguage#1#2#3#4#5{%
1733 \expandafter\ifx\csname lang@#1\endcsname\relax\else
1734 \global\expandafter\let\csname l@#1\endcsname
1735 \csname lang@#1\endcsname
1736 \fi}%
1737 \def\uselanguage#1{%
1738 \input language.def
1739 \endgroup
1740 \fi
1741 \fi
1742 \chardef\l@english\z@
1743 \fi

```

\addto It takes two arguments, a *<control sequence>* and T<sub>E</sub>X-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.

```

1744 \def\addto#1#2{%
1745   \ifx#1\@undefined
1746     \def#1{#2}%
1747   \else
1748     \ifx#1\relax
1749       \def#1{#2}%
1750     \else
1751       {\toks@\expandafter{#1#2}%
1752        \xdef#1{the\toks@}}%
1753   \fi
1754 \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?

```

1755 \def\bbl@withactive#1#2{%
1756   \begingroup
1757   \lccode`~=#2\relax
1758   \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  $\TeX$  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`.

```

1759 \def\bbl@redefine#1{%
1760   \edef\bbl@tempa{\bbl@stripslash#1}%
1761   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1762   \expandafter\def\csname\bbl@tempa\endcsname{
1763 \@onlypreamble\bbl@redefine

```

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

```

1764 \def\bbl@redefine@long#1{%
1765   \edef\bbl@tempa{\bbl@stripslash#1}%
1766   \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1767   \expandafter\long\expandafter\def\csname\bbl@tempa\endcsname{
1768 \@onlypreamble\bbl@redefine@long

```

`\bbl@redefineroobust` 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_`.

```

1769 \def\bbl@redefineroobust#1{%
1770   \edef\bbl@tempa{\bbl@stripslash#1}%
1771   \bbl@ifunset{\bbl@tempa\space}%
1772   {\expandafter\let\csname org@\bbl@tempa\endcsname#1%
1773    \bbl@exp{\def\#1{\protect\<\bbl@tempa\space>}}}%
1774   {\bbl@exp{\let\<org@\bbl@tempa>\<\bbl@tempa\space>}}}%
1775   \@namedef{\bbl@tempa\space}}
1776 \@onlypreamble\bbl@redefineroobust

```

### 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.

```

1777 \bbl@trace{Hooks}

```

```

1778 \newcommand\AddBabelHook[3][]{%
1779   \bbl@ifunset{bbl@hk@#2}{\EnableBabelHook{#2}}{}%
1780   \def\bbl@tempa##1,#3=##2,##3\@empty{\def\bbl@tempb{##2}}%
1781   \expandafter\bbl@tempa\bbl@evargs,#3=,\@empty
1782   \bbl@ifunset{bbl@ev@#2@#3@#1}%
1783     {\bbl@csarg\bbl@add{ev@#3@#1}{\bbl@elth{#2}}}%
1784     {\bbl@csarg\let{ev@#2@#3@#1}\relax}%
1785   \bbl@csarg\newcommand{ev@#2@#3@#1}[\bbl@tempb]}
1786 \newcommand\EnableBabelHook[1]{\bbl@csarg\let{hk@#1}\@firstofone}
1787 \newcommand\DisableBabelHook[1]{\bbl@csarg\let{hk@#1}\@gobble}
1788 \def\bbl@usehooks#1#2{%
1789   \def\bbl@elth##1{%
1790     \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#1}{#2}}%
1791     \bbl@cs{ev@#1}%
1792     \ifx\language\@undefined\else % Test required for Plain (?)
1793       \def\bbl@elth##1{%
1794         \bbl@cs{hk@##1}{\bbl@cl{ev@##1@#1}{#2}}%
1795         \bbl@cl{ev@#1}%
1796       \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).

```

1797 \def\bbl@evargs{% <- don't delete this comma
1798   everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
1799   adddialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
1800   beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
1801   hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%
1802   beforestart=0,language=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 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.

```

1803 \bbl@trace{Defining babelensure}
1804 \newcommand\babelensure[2][]{% TODO - revise test files
1805   \AddBabelHook{babel-ensure}{afterextras}{%
1806     \ifcase\bbl@select@type
1807       \bbl@cl{e}%
1808     \fi}%
1809   \begingroup
1810     \let\bbl@ens@include\@empty
1811     \let\bbl@ens@exclude\@empty
1812     \def\bbl@ens@fontenc{\relax}%
1813     \def\bbl@tempb##1{%
1814       \ifx\@empty##1\else\noexpand##1\expandafter\bbl@tempb\fi}%
1815     \edef\bbl@tempa{\bbl@tempb#1\@empty}%
1816     \def\bbl@tempb##1=##2\@{ \@namedef{bbl@ens@##1}{##2}}%
1817     \bbl@foreach\bbl@tempa{\bbl@tempb##1\@}%
1818     \def\bbl@tempc{\bbl@ensure}%
1819     \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1820       \expandafter{\bbl@ens@include}}%
1821     \expandafter\bbl@add\expandafter\bbl@tempc\expandafter{%
1822       \expandafter{\bbl@ens@exclude}}%

```

```

1823 \toks@\expandafter{\bbl@tempc}%
1824 \bbl@exp{%
1825 \endgroup
1826 \def\<bbl@e@#2>{\the\toks@{\bbl@ens@fontenc}}}}
1827 \def\bbl@ensure#1#2#3{% 1: include 2: exclude 3: fontenc
1828 \def\bbl@tempb##1{% elt for (excluding) \bbl@captionslist list
1829 \ifx##1\undefined % 3.32 - Don't assume the macro exists
1830 \edef##1{\noexpand\bbl@nocaption
1831 {\bbl@stripslash##1}{\language\language\bbl@stripslash##1}}}%
1832 \fi
1833 \ifx##1\@empty\else
1834 \in@{##1}{#2}%
1835 \ifin\else
1836 \bbl@ifunset{\bbl@ensure@\language\language}%
1837 {\bbl@exp{%
1838 \\\DeclareRobustCommand\<bbl@ensure@\language\language>[1]{%
1839 \\\foreignlanguage{\language\language}%
1840 {\ifx\relax#3\else
1841 \\\fontencoding{#3}\selectfont
1842 \fi
1843 #####1}}}}}%
1844 {}%
1845 \toks@\expandafter{##1}%
1846 \edef##1{%
1847 \bbl@csarg\noexpand{\ensure@\language\language}%
1848 {\the\toks@}}}%
1849 \fi
1850 \expandafter\bbl@tempb
1851 \fi}%
1852 \expandafter\bbl@tempb\bbl@captionslist\today\@empty
1853 \def\bbl@tempa##1{% elt for include list
1854 \ifx##1\@empty\else
1855 \bbl@csarg\in@{\ensure@\language\language\expandafter}\expandafter{##1}%
1856 \ifin\else
1857 \bbl@tempb##1\@empty
1858 \fi
1859 \expandafter\bbl@tempa
1860 \fi}%
1861 \bbl@tempa#1\@empty}
1862 \def\bbl@captionslist{%
1863 \prefacename\refname\abstractname\bibname\chaptername\appendixname
1864 \contentsname\listfigurename\listtablename\indexname\figurename
1865 \tablename\partname\enclname\ccname\headtoname\pagename\seename
1866 \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.

```

1867 \bbl@trace{Macro for setting language files up}
1868 \def\bbl@ldfinit{%
1869   \let\bbl@screset\@empty
1870   \let\BabelStrings\bbl@opt@string
1871   \let\BabelOptions\@empty
1872   \let\BabelLanguages\relax
1873   \ifx\originalTeX\@undefined
1874     \let\originalTeX\@empty
1875   \else
1876     \originalTeX
1877   \fi}
1878 \def\LdfInit#1#2{%
1879   \chardef\atcatcode=\catcode`\@
1880   \catcode`\@=11\relax
1881   \chardef\eqcatcode=\catcode`\=
1882   \catcode`\==12\relax
1883   \expandafter\if\expandafter\@backslashchar
1884     \expandafter\@car\string#2\@nil
1885   \ifx#2\@undefined\else
1886     \ldf@quit{#1}%
1887   \fi
1888 \else
1889   \expandafter\ifx\csname#2\endcsname\relax\else
1890     \ldf@quit{#1}%
1891   \fi
1892 \fi
1893 \bbl@ldfinit}

```

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

```

1894 \def\ldf@quit#1{%
1895   \expandafter\main@language\expandafter{#1}%
1896   \catcode`\@=\atcatcode \let\atcatcode\relax
1897   \catcode`\==\eqcatcode \let\eqcatcode\relax
1898   \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.

```

1899 \def\bbl@afterldf#1{% TODO. Merge into the next macro? Unused elsewhere
1900   \bbl@afterlang
1901   \let\bbl@afterlang\relax
1902   \let\BabelModifiers\relax
1903   \let\bbl@screset\relax}%
1904 \def\ldf@finish#1{%
1905   \ifx\loadlocalcfg\@undefined\else % For LaTeX 209
1906     \loadlocalcfg{#1}%
1907   \fi
1908   \bbl@afterldf{#1}%
1909   \expandafter\main@language\expandafter{#1}%
1910   \catcode`\@=\atcatcode \let\atcatcode\relax
1911   \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  $\LaTeX$ .

```
1912 \@onlypreamble\LdfInit
1913 \@onlypreamble\ldf@quit
1914 \@onlypreamble\ldf@finish
```

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

```
1915 \def\main@language#1{%
1916   \def\bbl@main@language{#1}%
1917   \let\language\name\bbl@main@language % TODO. Set localename
1918   \bbl@id@assign
1919   \bbl@patterns{\language}%}
```

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`.

```
1920 \def\bbl@beforestart{%
1921   \bbl@usehooks{beforestart}{}%
1922   \global\let\bbl@beforestart\relax}
1923 \AtBeginDocument{%
1924   \@nameuse{bbl@beforestart}%
1925   \if@filesw
1926     \providecommand\babel@aux[2]{}%
1927     \immediate\write\@mainaux{%
1928       \string\providecommand\string\babel@aux[2]{}%
1929       \immediate\write\@mainaux{\string\@nameuse{bbl@beforestart}}}%
1930   \fi
1931   \expandafter\selectlanguage\expandafter{\bbl@main@language}%
1932   \ifbbl@single % must go after the line above.
1933     \renewcommand\selectlanguage[1]{}%
1934     \renewcommand\foreignlanguage[2]{#2}%
1935     \global\let\babel@aux\@gobbletwo % Also as flag
1936   \fi
1937   \ifcase\bbl@engine\or\pagedir\bodydir\fi} % TODO - a better place
```

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

```
1938 \def\select@language@x#1{%
1939   \ifcase\bbl@select@type
1940     \bbl@ifsamestring\language\name{#1}{\select@language{#1}}%
1941   \else
1942     \select@language{#1}%
1943   \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.

```
1944 \bbl@trace{Shorthands}
1945 \def\bbl@add@special#1{% 1:a macro like \", \?, etc.
1946   \bbl@add\dospecials{\do#1}% test @sanitize = \relax, for back. compat.
1947   \bbl@ifunset{@sanitize}{\bbl@add@sanitize{\@makeother#1}}%
1948   \ifx\nfss@catcodes\@undefined\else % TODO - same for above
1949     \begingroup
```

```

1950      \catcode`#1\active
1951      \nfss@catcodes
1952      \ifnum\catcode`#1=\active
1953          \endgroup
1954          \bbl@add\nfss@catcodes{\@makeother#1}%
1955      \else
1956          \endgroup
1957      \fi
1958  \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.

```

1959 \def\bbl@remove@special#1{%
1960   \begingroup
1961     \def\x##1##2{\ifnum`#1=##2\noexpand\@empty
1962       \else\noexpand##1\noexpand##2\fi}%
1963     \def\do{\x\do}%
1964     \def\@makeother{\x\@makeother}%
1965   \edef\x{\endgroup
1966     \def\noexpand\dospecials{\dospecials}%
1967     \expandafter\ifx\csname @sanitize\endcsname\relax\else
1968       \def\noexpand\@sanitize{\@sanitize}%
1969     \fi}%
1970   \x}

```

`\initiate@active@char` A language definition file can call this macro to make a character active. This macro takes one argument, the character that is to be made active. When the character was already active this macro does nothing. Otherwise, this macro defines the control sequence `\normal@char` (*char*) to expand to the character in its ‘normal state’ and it defines the active character to expand to `\normal@char` (*char*) by default (*char* being the character to be made active). Later its definition can be changed to expand to `\active@char` (*char*) by calling `\bbl@activate{char}`. For example, to make the double quote character active one could have `\initiate@active@char{"}` in a language definition file. This defines " as `\active@prefix "active@char` (where the first " is the character with its original catcode, when the shorthand is created, and `\active@char` is a single token). In protected contexts, it expands to `\protect "` or `\noexpand "` (ie, with the original "); otherwise `\active@char` is executed. This macro in turn expands to `\normal@char` in “safe” contexts (eg, `\label`), but `\user@active` in normal “unsafe” ones. The latter search a definition in the user, language and system levels, in this order, but if none is found, `\normal@char` is used. However, a deactivated shorthand (with `\bbl@deactivate` is defined as `\active@prefix "\normal@char`).

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

```

1971 \def\bbl@active@def#1#2#3#4{%
1972   \@namedef{#3#1}{%
1973     \expandafter\ifx\csname#2@sh@#1\endcsname\relax
1974       \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
1975     \else
1976       \bbl@afterfi\csname#2@sh@#1\endcsname
1977     \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.

```

1978   \long\@namedef{#3@arg#1}##1{%
1979     \expandafter\ifx\csname#2@sh@#1\string##1\endcsname\relax
1980       \bbl@afterelse\csname#4#1\endcsname##1%
1981     \else
1982       \bbl@afterfi\csname#2@sh@#1\string##1\endcsname
1983     \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.

```

1984 \def\initiate@active@char#1{%
1985   \bbl@ifunset{active@char\string#1}%
1986   {\bbl@withactive
1987     {\expandafter\@initiate@active@char\expandafter}#1\string#1}%
1988   {}}

```

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 treatment to avoid making them \relax).

```

1989 \def\@initiate@active@char#1#2#3{%
1990   \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
1991   \ifx#1\@undefined
1992     \bbl@csarg\edef{oridef@#2}{\let\noexpand#1\noexpand\@undefined}%
1993   \else
1994     \bbl@csarg\let{oridef@#2}#1%
1995     \bbl@csarg\edef{oridef@#2}{%
1996       \let\noexpand#1%
1997       \expandafter\noexpand\csname bbl@oridef@#2\endcsname}%
1998   \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).

```

1999   \ifx#1#3\relax
2000     \expandafter\let\csname normal@char#2\endcsname#3%
2001   \else
2002     \bbl@info{Making #2 an active character}%
2003     \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
2004       \@namedef{normal@char#2}{%
2005         \textormath{#3}{\csname bbl@oridef@#2\endcsname}}%
2006     \else
2007       \@namedef{normal@char#2}{#3}%
2008     \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).

```

2009   \bbl@restoreactive{#2}%
2010   \AtBeginDocument{%
2011     \catcode`#2\active
2012     \if@filesw
2013       \immediate\write\@mainaux{\catcode`\string#2\active}%
2014     \fi}%
2015   \expandafter\bbl@add@special\csname#2\endcsname
2016   \catcode`#2\active
2017 \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>).

```

2018 \let\bbl@tempa\@firstoftwo
2019 \if\string^#2%
2020 \def\bbl@tempa{\noexpand\textormath}%
2021 \else
2022 \ifx\bbl@mathnormal\@undefined\else
2023 \let\bbl@tempa\bbl@mathnormal
2024 \fi
2025 \fi
2026 \expandafter\edef\csname active@char#2\endcsname{%
2027 \bbl@tempa
2028 {\noexpand\if@safe@actives
2029 \noexpand\expandafter
2030 \expandafter\noexpand\csname normal@char#2\endcsname
2031 \noexpand\else
2032 \noexpand\expandafter
2033 \expandafter\noexpand\csname bbl@doactive#2\endcsname
2034 \noexpand\fi}%
2035 {\expandafter\noexpand\csname normal@char#2\endcsname}}%
2036 \bbl@csarg\edef{doactive#2}{%
2037 \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

`\active@prefix <char> \normal@char<char>`

(where `\active@char<char>` is *one* control sequence!).

```

2038 \bbl@csarg\edef{active@#2}{%
2039 \noexpand\active@prefix\noexpand#1%
2040 \expandafter\noexpand\csname active@char#2\endcsname}%
2041 \bbl@csarg\edef{normal@#2}{%
2042 \noexpand\active@prefix\noexpand#1%
2043 \expandafter\noexpand\csname normal@char#2\endcsname}%
2044 \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.

```

2045 \bbl@active@def#2\user@group{user@active}{language@active}%
2046 \bbl@active@def#2\language@group{language@active}{system@active}%
2047 \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.

```

2048 \expandafter\edef\csname\user@group @sh@#2@\endcsname
2049 {\expandafter\noexpand\csname normal@char#2\endcsname}%
2050 \expandafter\edef\csname\user@group @sh@#2@\string\protect\endcsname
2051 {\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.

```

2052 \if\string'#2%
2053 \let\prim@s\bbl@prim@s
2054 \let\active@math@prime#1%

```



```

2055 \fi
2056 \bbl@usehooks{initiateactive}{\#1}{\#2}{\#3}}

```

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

```

2057 <<(*More package options)>> ≡
2058 \DeclareOption{math=active}{}
2059 \DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
2060 <</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* the end of the ldf.

```

2061 \@ifpackagewith{babel}{KeepShorthandsActive}%
2062 {\let\bbl@restoreactive\@gobble}%
2063 {\def\bbl@restoreactive#1{%
2064   \bbl@exp{%
2065     \\\AfterBabelLanguage\\CurrentOption
2066     {\catcode`#1=\the\catcode`#1\relax}%
2067     \\\AtEndOfPackage
2068     {\catcode`#1=\the\catcode`#1\relax}}}%
2069   \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.

```

2070 \def\bbl@sh@select#1#2{%
2071   \expandafter\ifx\csname#1@sh@#2@sel\endcsname\relax
2072     \bbl@afterelse\bbl@scndcs
2073   \else
2074     \bbl@afterfi\csname#1@sh@#2@sel\endcsname
2075   \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.

```

2076 \begingroup
2077 \bbl@ifunset{ifincsname}% TODO. Ugly. Correct?
2078 {\gdef\active@prefix#1{%
2079   \ifx\protect\@typeset@protect
2080   \else
2081     \ifx\protect\@unexpandable@protect
2082       \noexpand#1%
2083     \else
2084       \protect#1%
2085     \fi
2086     \expandafter\@gobble
2087   \fi}}
2088 {\gdef\active@prefix#1{%
2089   \ifincsname
2090     \string#1%
2091     \expandafter\@gobble
2092   \else
2093     \ifx\protect\@typeset@protect
2094     \else

```

```

2095      \ifx\protect\@unexpandable@protect
2096      \noexpand#1%
2097      \else
2098      \protect#1%
2099      \fi
2100      \expandafter\expandafter\expandafter\@gobble
2101      \fi
2102      \fi}}
2103 \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>.

2104 \newif\if@safe@actives
2105 \@safe@activesfalse

\bbbl@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.

2106 \def\bbbl@restore@actives{\if@safe@actives\@safe@activesfalse\fi}

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

2107 \def\bbbl@activate#1{%
2108   \bbbl@withactive{\expandafter\let\expandafter}#1%
2109   \csname bbl@active@\string#1\endcsname}
2110 \def\bbbl@deactivate#1{%
2111   \bbbl@withactive{\expandafter\let\expandafter}#1%
2112   \csname bbl@normal@\string#1\endcsname}

\bbbl@firstcs These macros are used only as a trick when declaring shorthands.
\bbbl@scndcs
2113 \def\bbbl@firstcs#1#2{\csname#1\endcsname}
2114 \def\bbbl@scndcs#1#2{\csname#2\endcsname}

\declare@shorthand The command \declare@shorthand is used to declare a shorthand on a certain level. It takes three
arguments:
1. a name for the collection of shorthands, i.e. ‘system’, or ‘dutch’;
2. the character (sequence) that makes up the shorthand, i.e. ~ or "a;
3. the code to be executed when the shorthand is encountered.

The auxiliary macro \babel@texpdf improves the interoperativity with hyperref and takes 4
arguments: (1) The TEX code in text mode, (2) the string for hyperref, (3) the TEX code in math mode,
and (4), which is currently ignored, but it's meant for a string in math mode, like a minus sign instead
of an hyphen (currently hyperref doesn't discriminate the mode). This macro may be used in lateX
files.

2115 \def\babel@texpdf#1#2#3#4{%
2116   \ifx\texorpdfstring\@undefined
2117     \textormath{#1}{#2}%
2118   \else
2119     \texorpdfstring{\textormath{#1}{#3}}{#2}%
2120     % \texorpdfstring{\textormath{#1}{#3}}{\textormath{#2}{#4}}%
2121   \fi}
2122 %
2123 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
2124 \def\@decl@short#1#2#3\@nil#4{%
2125   \def\bbbl@tempa{#3}%

```

```

2126 \ifx\bb1@tempa\@empty
2127 \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bb1@scndcs
2128 \bb1@ifunset{#1@sh@\string#2@}{}%
2129 {\def\bb1@tempa{#4}%
2130 \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bb1@tempa
2131 \else
2132 \bb1@info
2133 {Redefining #1 shorthand \string#2\\%
2134 in language \CurrentOption}%
2135 \fi}%
2136 \@namedef{#1@sh@\string#2@}{#4}%
2137 \else
2138 \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bb1@firstcs
2139 \bb1@ifunset{#1@sh@\string#2@\string#3@}{}%
2140 {\def\bb1@tempa{#4}%
2141 \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bb1@tempa
2142 \else
2143 \bb1@info
2144 {Redefining #1 shorthand \string#2\string#3\\%
2145 in language \CurrentOption}%
2146 \fi}%
2147 \@namedef{#1@sh@\string#2@\string#3@}{#4}%
2148 \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.

```

2149 \def\textormath{%
2150 \ifmmode
2151 \expandafter\@secondoftwo
2152 \else
2153 \expandafter\@firstoftwo
2154 \fi}

```

`\user@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’.

```

2155 \def\user@group{user}
2156 \def\language@group{english} % TODO. I don't like defaults
2157 \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.

```

2158 \def\useshorthands{%
2159 \@ifstar\bb1@usesh@s{\bb1@usesh@x{}}
2160 \def\bb1@usesh@s#1{%
2161 \bb1@usesh@x
2162 {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bb1@activate{#1}}}%
2163 {#1}}
2164 \def\bb1@usesh@x#1#2{%
2165 \bb1@ifshorthand{#2}%
2166 {\def\user@group{user}%
2167 \initiate@active@char{#2}%
2168 #1%
2169 \bb1@activate{#2}}%
2170 {\bb1@error
2171 {Cannot declare a shorthand turned off (\string#2)}
2172 {Sorry, but you cannot use shorthands which have been\\%
2173 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.

```

2174 \def\user@language@group{user@\language@group}
2175 \def\bbl@set@user@generic#1#2{%
2176   \bbl@ifunset{user@generic@active#1}%
2177   {\bbl@active@def#1\user@language@group{user@active}{user@generic@active}%
2178    \bbl@active@def#1\user@group{user@generic@active}{language@active}%
2179    \expandafter\edef\csname#2@sh@#1@@\endcsname{%
2180      \expandafter\noexpand\csname normal@char#1\endcsname}%
2181      \expandafter\edef\csname#2@sh@#1@\string\protect@\endcsname{%
2182        \expandafter\noexpand\csname user@active#1\endcsname}}%
2183   \@empty}
2184 \newcommand\defineshorthand[3][user]{%
2185   \edef\bbl@tempa{\zap@space#1 \@empty}%
2186   \bbl@for\bbl@tempb\bbl@tempa{%
2187     \if*\expandafter\@car\bbl@tempb\@nil
2188       \edef\bbl@tempb{user@\expandafter\@gobble\bbl@tempb}%
2189       \@expandtwoargs
2190       \bbl@set@user@generic{\expandafter\string\@car#2\@nil}\bbl@tempb
2191     \fi
2192     \declare@shorthand{\bbl@tempb}{#2}{#3}}

```

`\languageshorthands` A user level command to change the language from which shorthands are used. Unfortunately, babel currently does not keep track of defined groups, and therefore there is no way to catch a possible change in casing to fix it in the same way languages names are fixed. [TODO].

```

2193 \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 latest to `\active@char`.

```

2194 \def\aliasshorthand#1#2{%
2195   \bbl@ifshorthand{#2}%
2196   {\expandafter\ifx\csname active@char\string#2\endcsname\relax
2197     \ifx\document\@notprerr
2198       \@notshorthand{#2}%
2199     \else
2200       \initiate@active@char{#2}%
2201       \expandafter\let\csname active@char\string#2\expandafter\endcsname
2202         \csname active@char\string#1\endcsname
2203       \expandafter\let\csname normal@char\string#2\expandafter\endcsname
2204         \csname normal@char\string#1\endcsname
2205       \bbl@activate{#2}%
2206     \fi
2207   \fi}%
2208   {\bbl@error
2209     {Cannot declare a shorthand turned off (\string#2)}
2210     {Sorry, but you cannot use shorthands which have been\\
2211      turned off in the package options}}}

```

`\@notshorthand`

```

2212 \def\@notshorthand#1{%
2213   \bbl@error{%
2214     The character '\string #1' should be made a shorthand character;\\
2215     add the command \string\usesshorthands\string{#1\string} to
2216     the preamble.\\

```

```

2217     I will ignore your instruction}%
2218     {You may proceed, but expect unexpected results}}

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

2219 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
2220 \DeclareRobustCommand*\shorthandoff{%
2221   \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
2222 \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.

2223 \def\bbl@switch@sh#1#2{%
2224   \ifx#2\@nnil\else
2225     \bbl@ifunset{\bbl@active@\string#2}%
2226     {\bbl@error
2227       {I cannot switch '\string#2' on or off--not a shorthand}%
2228       {This character is not a shorthand. Maybe you made\%
2229         a typing mistake? I will ignore your instruction}}}%
2230     {\ifcase#1%
2231       \catcode`#212\relax
2232       \or
2233       \catcode`#2\active
2234       \or
2235       \csname bbl@oricat@\string#2\endcsname
2236       \csname bbl@oridef@\string#2\endcsname
2237       \fi}%
2238     \bbl@afterfi\bbl@switch@sh#1%
2239   \fi}

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

2240 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
2241 \def\bbl@putsh#1{%
2242   \bbl@ifunset{\bbl@active@\string#1}%
2243   {\bbl@putsh@i#1\@empty\@nnil}%
2244   {\csname bbl@active@\string#1\endcsname}}
2245 \def\bbl@putsh@i#1#2\@nnil{%
2246   \csname\language@group @sh@\string#1@%
2247     \ifx\@empty#2\else\string#2@\fi\endcsname}
2248 \ifx\bbl@opt@shorthands\@nnil\else
2249   \let\bbl@s@initiate@active@char\initiate@active@char
2250   \def\initiate@active@char#1{%
2251     \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
2252   \let\bbl@s@switch@sh\bbl@switch@sh
2253   \def\bbl@switch@sh#1#2{%
2254     \ifx#2\@nnil\else
2255       \bbl@afterfi
2256       \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
2257     \fi}
2258   \let\bbl@s@activate\bbl@activate
2259   \def\bbl@activate#1{%
2260     \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
2261   \let\bbl@s@deactivate\bbl@deactivate

```

```

2262 \def\bbld@deactivate#1{%
2263   \bbld@ifshorthand{#1}{\bbld@s@deactivate{#1}}{}}
2264 \fi

```

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

```

2265 \newcommand\ifbabelshorthand[3]{\bbld@ifunset{bbld@active@string#1}{#3}{#2}}

```

`\bbld@prim@s` One of the internal macros that are involved in substituting `\prime` for each right quote in 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.

```

2266 \def\bbld@prim@s{%
2267   \prime\futurelet\@let@token\bbld@pr@m@s}
2268 \def\bbld@if@primes#1#2{%
2269   \ifx#1\@let@token
2270     \expandafter\@firstoftwo
2271   \else\ifx#2\@let@token
2272     \bbld@afterelse\expandafter\@firstoftwo
2273   \else
2274     \bbld@afterfi\expandafter\@secondoftwo
2275   \fi\fi}
2276 \begingroup
2277   \catcode`\^=7 \catcode`\*=\active \lccode`\*=\^
2278   \catcode`\'=12 \catcode`\"=\active \lccode`\"=\'
2279   \lowercase{%
2280     \gdef\bbld@pr@m@s{%
2281       \bbld@if@primes"'"%
2282       \pr@@@s
2283       {\bbld@if@primes*\^*\pr@@@t\egroup}}
2284 \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).

```

2285 \initiate@active@char{~}
2286 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
2287 \bbld@activate{~}

```

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

```

2288 \expandafter\def\csname OT1dqpos\endcsname{127}
2289 \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

```

2290 \ifx\f@encoding\undefined
2291   \def\f@encoding{OT1}
2292 \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.

```

2293 \bbl@trace{Language attributes}
2294 \newcommand\languageattribute[2]{%
2295   \def\bbl@tempc{#1}%
2296   \bbl@fixname\bbl@tempc
2297   \bbl@iflanguage\bbl@tempc{%
2298     \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@attrs`. When that control sequence is not yet defined this attribute is certainly not selected before.

```

2299     \ifx\bbl@known@attrs\undefined
2300       \in@false
2301     \else
2302       \bbl@xin@{,\bbl@tempc-##1,}{,\bbl@known@attrs,}%
2303     \fi
2304     \ifin@
2305       \bbl@warning{%
2306         You have more than once selected the attribute '##1'\%
2307         for language #1. Reported}%
2308     \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.

```

2309       \bbl@exp{%
2310         \\bbl@add@list\\bbl@known@attrs{\bbl@tempc-##1}}%
2311       \edef\bbl@tempa{\bbl@tempc-##1}%
2312       \expandafter\bbl@ifknown@trib\expandafter{\bbl@tempa}\bbl@attributes%
2313       {\csname\bbl@tempc @attr##1\endcsname}%
2314       {\@attrerr{\bbl@tempc}{##1}}%
2315     \fi}}}
2316 \@onlypreamble\languageattribute

```

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

```

2317 \newcommand*{\@attrerr}[2]{%
2318   \bbl@error
2319   {The attribute #2 is unknown for language #1.}%
2320   {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}`.

```

2321 \def\bbl@declare@ttribute#1#2#3{%
2322   \bbl@xin@{,#2,}{,\BabelModifiers,}%
2323   \ifin@
2324     \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
2325   \fi
2326   \bbl@add@list\bbl@attributes{#1-#2}%
2327   \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.

```

2328 \def\bbl@ifattributeset#1#2#3#4{%
2329   \ifx\bbl@known@attribs\@undefined
2330     \in@false
2331   \else
2332     \bbl@xin@{,#1-#2,}{,\bbl@known@attribs,}%
2333   \fi
2334   \ifin@
2335     \bbl@afterelse#3%
2336   \else
2337     \bbl@afterfi#4%
2338   \fi}

```

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

```

2339 \def\bbl@ifknown@ttrib#1#2{%
2340   \let\bbl@tempa\@secondoftwo
2341   \bbl@loopx\bbl@tempb{#2}{%
2342     \expandafter\in@\expandafter{\expandafter,\bbl@tempb,}{,#1,}%
2343   \ifin@
2344     \let\bbl@tempa\@firstoftwo
2345   \else
2346   \fi}%
2347   \bbl@tempa}

```

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

```

2348 \def\bbl@clear@ttribs{%
2349   \ifx\bbl@attributes\@undefined\else
2350     \bbl@loopx\bbl@tempa{\bbl@attributes}{%
2351       \expandafter\bbl@clear@ttrib\bbl@tempa.
2352     }%
2353     \let\bbl@attributes\@undefined
2354   \fi}
2355 \def\bbl@clear@ttrib#1-#2.{%
2356   \expandafter\let\csname#1@attr@#2\endcsname\@undefined}
2357 \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`

```

2358 \bbl@trace{Macros for saving definitions}
2359 \def\babel@beginsave{\babel@savecnt\z@}

```

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

```

2360 \newcount\babel@savecnt
2361 \babel@beginsave

```



`\babel@save` The macro `\babel@save⟨csname⟩` saves the current meaning of the control sequence `⟨csname⟩` to `\originalTeX`<sup>32</sup>. 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.

```
2362 \def\babel@save#1{%
2363   \expandafter\let\csname babel@number\babel@savecnt\endcsname#1\relax
2364   \toks@\expandafter{\originalTeX\let#1=}%
2365   \bbl@exp{%
2366     \def\originalTeX{\the\toks@\<babel@number\babel@savecnt>\relax}}%
2367   \advance\babel@savecnt\@ne}
2368 \def\babel@savevariable#1{%
2369   \toks@\expandafter{\originalTeX #1}%
2370   \bbl@exp{\def\originalTeX{\the\toks@\the#1\relax}}}
```

`\bbl@frenchspacing` Some languages need to have `\frenchspacing` in effect. Others don't want that. The command `\bbl@frenchspacing` switches it on when it isn't already in effect and `\bbl@nonfrenchspacing` switches it off if necessary. A more refined way to switch the catcodes is done with ini files. Here an auxiliary macro is defined, but the main part is in `\babelprovide`. This new method should be ideally the default one.

```
2371 \def\bbl@frenchspacing{%
2372   \ifnum\the\sfcode`\.=\@m
2373     \let\bbl@nonfrenchspacing\relax
2374   \else
2375     \frenchspacing
2376     \let\bbl@nonfrenchspacing\nonfrenchspacing
2377   \fi}
2378 \let\bbl@nonfrenchspacing\nonfrenchspacing
2379 \let\bbl@elt\relax
2380 \edef\bbl@fs@chars{%
2381   \bbl@elt{\string.}\@m{3000}\bbl@elt{\string?}\@m{3000}%
2382   \bbl@elt{\string!}\@m{3000}\bbl@elt{\string:}\@m{2000}%
2383   \bbl@elt{\string;}\@m{1500}\bbl@elt{\string,}\@m{1250}}
```

## 9.8 Short tags

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

```
2384 \bbl@trace{Short tags}
2385 \def\babeltags#1{%
2386   \edef\bbl@tempa{\zap@space#1 \@empty}%
2387   \def\bbl@tempb##1=##2\@{%
2388     \edef\bbl@tempc{%
2389       \noexpand\newcommand
2390       \expandafter\noexpand\csname ##1\endcsname{%
2391         \noexpand\protect
2392         \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}
2393       \noexpand\newcommand
2394       \expandafter\noexpand\csname text##1\endcsname{%
2395         \noexpand\foreignlanguage{##2}}
2396       \bbl@tempc}%
2397   \bbl@for\bbl@tempa\bbl@tempa{%
2398     \expandafter\bbl@tempb\bbl@tempa\@{}}
```

<sup>32</sup>`\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.

```

2399 \bbl@trace{Hyphens}
2400 \@onlypreamble\babelhyphenation
2401 \AtEndOfPackage{%
2402   \newcommand\babelhyphenation[2][\@empty]{%
2403     \ifx\bbl@hyphenation@relax
2404       \let\bbl@hyphenation@\@empty
2405     \fi
2406     \ifx\bbl@hyphlist\@empty\else
2407       \bbl@warning{%
2408         You must not intermingle \string\selectlanguage\space and\\%
2409         \string\babelhyphenation\space or some exceptions will not\\%
2410         be taken into account. Reported}%
2411       \fi
2412       \ifx\@empty#1%
2413         \protected@edef\bbl@hyphenation@{\bbl@hyphenation@\space#2}%
2414       \else
2415         \bbl@vforeach{#1}{%
2416           \def\bbl@tempa{##1}%
2417           \bbl@fixname\bbl@tempa
2418           \bbl@iflanguage\bbl@tempa{%
2419             \bbl@csarg\protected@edef{hyphenation@\bbl@tempa}{%
2420               \bbl@ifunset{bbl@hyphenation@\bbl@tempa}%
2421               {}%
2422               {\csname bbl@hyphenation@\bbl@tempa\endcsname\space}%
2423               #2}}}%
2424         \fi}}

```

`\bbl@allowhyphens` This macro makes hyphenation possible. Basically its definition is nothing more than `\nobreak` `\hskip 0pt plus 0pt`<sup>33</sup>.

```

2425 \def\bbl@allowhyphens{\ifvmode\else\nobreak\hskip\z@skip\fi}
2426 \def\bbl@t@one{T1}
2427 \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`.

```

2428 \newcommand\babelnullhyphen{\char\hyphenchar\font}
2429 \def\babelhyphen{\active@prefix\babelhyphen\bbl@hyphen}
2430 \def\bbl@hyphen{%
2431   \@ifstar{\bbl@hyphen@i @}{\bbl@hyphen@i \@empty}}
2432 \def\bbl@hyphen@i#1#2{%
2433   \bbl@ifunset{bbl@hy@#1#2\@empty}%
2434   {\csname bbl@#1usehyphen\endcsname{\discretionary{#2}{}{#2}}}%
2435   {\csname bbl@hy@#1#2\@empty\endcsname}}

```

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

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

<sup>33</sup> $\TeX$  begins and ends a word for hyphenation at a glue node. The penalty prevents a linebreak at this glue node.

```

2436 \def\bbl@usehyphen#1{%
2437   \leavevmode
2438   \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
2439   \nobreak\hskip\z@skip}
2440 \def\bbl@usehyphen#1{%
2441   \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}

```

The following macro inserts the hyphen char.

```

2442 \def\bbl@hyphenchar{%
2443   \ifnum\hyphenchar\font=\m@ne
2444     \babeinullhyphen
2445   \else
2446     \char\hyphenchar\font
2447   \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.

```

2448 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
2449 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
2450 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
2451 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
2452 \def\bbl@hy@nobreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}}
2453 \def\bbl@hy@nobreak{\mbox{\bbl@hyphenchar}}
2454 \def\bbl@hy@repeat{%
2455   \bbl@usehyphen{%
2456     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
2457 \def\bbl@hy@repeat{%
2458   \bbl@usehyphen{%
2459     \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}}
2460 \def\bbl@hy@empty{\hskip\z@skip}
2461 \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.

```

2462 \def\bbl@disc#1#2{\nobreak\discretionary{#2-}{#1}\bbl@allowhyphens}

```

## 9.10 Multiencoding strings

The aim following commands is to provide a common 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.

```

2463 \bbl@trace{Multiencoding strings}
2464 \def\bbl@tglobal#1{\global\let#1#1}
2465 \def\bbl@recatcode#1{% TODO. Used only once?
2466   \@tempcnta="7F
2467   \def\bbl@tempa{%
2468     \ifnum\@tempcnta>"FF\else
2469       \catcode\@tempcnta=#1\relax
2470       \advance\@tempcnta\@ne
2471       \expandafter\bbl@tempa
2472     \fi}%
2473   \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).

```
2474 \@ifpackagewith{babel}{nocase}%
2475   {\let\bbl@patchuclc\relax}%
2476   {\def\bbl@patchuclc{%
2477     \global\let\bbl@patchuclc\relax
2478     \g@addto@macro\@uclclist{\reserved@b{\reserved@b\bbl@uclc}}%
2479     \gdef\bbl@uclc##1{%
2480       \let\bbl@encoded\bbl@encoded@uclc
2481       \bbl@ifunset{\language @bbl@uclc}% and resumes it
2482       {##1}%
2483       {\let\bbl@tempa##1\relax % Used by LANG@bbl@uclc
2484         \csname\language @bbl@uclc\endcsname}%
2485       {\bbl@tolower\@empty}{\bbl@toupper\@empty}}}%
2486     \gdef\bbl@tolower{\csname\language @bbl@lc\endcsname}%
2487     \gdef\bbl@toupper{\csname\language @bbl@uc\endcsname}}%
2488 <<(*More package options)>> ≡
2489 \DeclareOption{nocase}{}
2490 <</More package options>>
```

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

```
2491 <<(*More package options)>> ≡
2492 \let\bbl@opt@strings\@nnil % accept strings=value
2493 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
2494 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
2495 \def\BabelStringsDefault{generic}
2496 <</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.

```
2497 \@onlypreamble\StartBabelCommands
2498 \def\StartBabelCommands{%
2499   \begingroup
2500   \bbl@recatcode{11}%
2501   <<Macros local to BabelCommands>>
2502   \def\bbl@provstring##1##2{%
2503     \providecommand##1{##2}%
2504     \bbl@tglobal##1}%
2505   \global\let\bbl@scafter\@empty
2506   \let\StartBabelCommands\bbl@startcmds
2507   \ifx\BabelLanguages\relax
2508     \let\BabelLanguages\CurrentOption
2509   \fi
2510   \begingroup
2511   \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
2512   \StartBabelCommands}
2513 \def\bbl@startcmds{%
2514   \ifx\bbl@screset\@nnil\else
2515     \bbl@usehooks{stopcommands}{}%
2516   \fi
```

```

2517 \endgroup
2518 \begingroup
2519 \@ifstar
2520 {\ifx\bbbl@opt@strings\@nnil
2521 \let\bbbl@opt@strings\BabelStringsDefault
2522 \fi
2523 \bbbl@startcmds@i}%
2524 \bbbl@startcmds@i}
2525 \def\bbbl@startcmds@i#1#2{%
2526 \edef\bbbl@L{\zap@space#1 \@empty}%
2527 \edef\bbbl@G{\zap@space#2 \@empty}%
2528 \bbbl@startcmds@ii}
2529 \let\bbbl@startcommands\StartBabelCommands

```

Parse the encoding info to get the label, input, and font parts.

Select the behavior of \SetString. There 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.

```

2530 \newcommand\bbbl@startcmds@ii[1][\@empty]{%
2531 \let\SetString@gobbletwo
2532 \let\bbbl@stringdef@gobbletwo
2533 \let\AfterBabelCommands@gobble
2534 \ifx\@empty#1%
2535 \def\bbbl@sc@label{generic}%
2536 \def\bbbl@encstring##1##2{%
2537 \ProvideTextCommandDefault##1{##2}%
2538 \bbbl@toglobal##1%
2539 \expandafter\bbbl@toglobal\csname\string?\string##1\endcsname}%
2540 \let\bbbl@sctest\in@true
2541 \else
2542 \let\bbbl@sc@charset\space % <- zapped below
2543 \let\bbbl@sc@fontenc\space % <- " "
2544 \def\bbbl@tempa##1=##2\@nil{%
2545 \bbbl@csarg\edef{sc@\zap@space##1 \@empty}{##2 }}%
2546 \bbbl@foreach{label=#1}{\bbbl@tempa##1\@nil}%
2547 \def\bbbl@tempa##1 ##2{% space -> comma
2548 ##1%
2549 \ifx\@empty##2\else\ifx,##1,\else,\fi\bbbl@afterfi\bbbl@tempa##2\fi}%
2550 \edef\bbbl@sc@fontenc{\expandafter\bbbl@tempa\bbbl@sc@fontenc\@empty}%
2551 \edef\bbbl@sc@label{\expandafter\zap@space\bbbl@sc@label\@empty}%
2552 \edef\bbbl@sc@charset{\expandafter\zap@space\bbbl@sc@charset\@empty}%
2553 \def\bbbl@encstring##1##2{%
2554 \bbbl@foreach\bbbl@sc@fontenc{%
2555 \bbbl@ifunset{T@###1}%
2556 {}%
2557 {\ProvideTextCommand##1{####1}{##2}%
2558 \bbbl@toglobal##1%
2559 \expandafter
2560 \bbbl@toglobal\csname####1\string##1\endcsname}}}%
2561 \def\bbbl@sctest{%
2562 \bbbl@xin@{\bbbl@opt@strings,}{\bbbl@sc@label,\bbbl@sc@fontenc,}}%
2563 \fi
2564 \ifx\bbbl@opt@strings\@nnil % ie, no strings key -> defaults
2565 \else\ifx\bbbl@opt@strings\relax % ie, strings=encoded

```

```

2566 \let\AfterBabelCommands\bbbl@aftercmds
2567 \let\SetString\bbbl@setstring
2568 \let\bbbl@stringdef\bbbl@encstring
2569 \else % ie, strings=value
2570 \bbbl@scstest
2571 \ifin@
2572 \let\AfterBabelCommands\bbbl@aftercmds
2573 \let\SetString\bbbl@setstring
2574 \let\bbbl@stringdef\bbbl@provstring
2575 \fi\fi\fi
2576 \bbbl@scswitch
2577 \ifx\bbbl@G\@empty
2578 \def\SetString##1##2{%
2579 \bbbl@error{Missing group for string \string##1}%
2580 {You must assign strings to some category, typically\\%
2581 captions or extras, but you set none}}%
2582 \fi
2583 \ifx\@empty#1%
2584 \bbbl@usehooks{defaultcommands}{}%
2585 \else
2586 \@expandtwoargs
2587 \bbbl@usehooks{encodedcommands}{\bbbl@sc@charset}{\bbbl@sc@fontenc}}%
2588 \fi}

```

There are two versions of `\bbbl@scswitch`. The first version is used when ldfs are read, and it makes sure `\langle group \rangle \langle language \rangle` is reset, but only once (`\bbbl@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 `\bbbl@forlang` loops `\bbbl@L` but its body is executed only if the value is in `\BabelLanguages` (inside babel) or `\date \langle language \rangle` is defined (after babel has been loaded). There are also two version of `\bbbl@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).

```

2589 \def\bbbl@forlang#1#2{%
2590 \bbbl@for#1\bbbl@L{%
2591 \bbbl@xin@{, #1,}{, \BabelLanguages,}%
2592 \ifin@#2\relax\fi}}
2593 \def\bbbl@scswitch{%
2594 \bbbl@forlang\bbbl@tempa{%
2595 \ifx\bbbl@G\@empty\else
2596 \ifx\SetString\@gobbletwo\else
2597 \edef\bbbl@GL{\bbbl@G\bbbl@tempa}%
2598 \bbbl@xin@{, \bbbl@GL,}{, \bbbl@screset,}%
2599 \ifin@\else
2600 \global\expandafter\let\csname\bbbl@GL\endcsname\@undefined
2601 \xdef\bbbl@screset{\bbbl@screset, \bbbl@GL}%
2602 \fi
2603 \fi
2604 \fi}}
2605 \AtEndOfPackage{%
2606 \def\bbbl@forlang#1#2{\bbbl@for#1\bbbl@L{\bbbl@ifunset{date#1}{}{#2}}}%
2607 \let\bbbl@scswitch\relax}
2608 \onlypreamble\EndBabelCommands
2609 \def\EndBabelCommands{%
2610 \bbbl@usehooks{stopcommands}{}%
2611 \endgroup
2612 \endgroup
2613 \bbbl@scafter}
2614 \let\bbbl@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 `\providescommand`). 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.

```

2615 \def\bbl@setstring#1#2{% eg, \prefacename{<string>}
2616   \bbl@forlang\bbl@tempa{%
2617     \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
2618     \bbl@ifunset{\bbl@LC}% eg, \germanchaptername
2619       {\bbl@exp{%
2620         \global\bbbl@add\<\bbl@G\bbl@tempa>{\bbbl@scset\#1\<\bbl@LC>}}}%
2621       }%
2622     \def\BabelString{#2}%
2623     \bbl@usehooks{stringprocess}{}%
2624     \expandafter\bbl@stringdef
2625     \csname\bbl@LC\expandafter\endcsname\expandafter{\BabelString}}}
```

Now, some additional 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`.

```

2626 \ifx\bbl@opt@strings\relax
2627   \def\bbl@scset#1#2{\def#1{\bbl@encoded#2}}
2628   \bbl@patchuclc
2629   \let\bbl@encoded\relax
2630   \def\bbl@encoded@uclc#1{%
2631     \@inmathwarn#1%
2632     \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
2633       \expandafter\ifx\csname ?\string#1\endcsname\relax
2634         \TextSymbolUnavailable#1%
2635       \else
2636         \csname ?\string#1\endcsname
2637       \fi
2638     \else
2639       \csname\cf@encoding\string#1\endcsname
2640     \fi}
2641 \else
2642   \def\bbl@scset#1#2{\def#1{#2}}
2643 \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.

```

2644 <<*Macros local to BabelCommands>> ≡
2645 \def\SetStringLoop##1##2{%
2646   \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
2647   \count@\z@
2648   \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
2649     \advance\count@\@ne
2650     \toks@\expandafter{\bbl@tempa}%
2651     \bbl@exp{%
2652       \\SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%
2653       \count@=\the\count@\relax}}}%
2654 <</Macros local to BabelCommands>>
```

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

```

2655 \def\bbl@aftercmds#1{%
2656   \toks@\expandafter{\bbl@scafter#1}%
2657   \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.

```
2658 <<*Macros local to BabelCommands>> ≡
2659 \newcommand\SetCase[3][]{%
2660   \bbl@patchuclc
2661   \bbl@forlang\bbl@tempa{%
2662     \expandafter\bbl@encstring
2663     \csname\bbl@tempa @bbl@uclc\endcsname{\bbl@tempa##1}%
2664     \expandafter\bbl@encstring
2665     \csname\bbl@tempa @bbl@uc\endcsname{##2}%
2666     \expandafter\bbl@encstring
2667     \csname\bbl@tempa @bbl@lc\endcsname{##3}}}%
2668 <</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.

```
2669 <<*Macros local to BabelCommands>> ≡
2670 \newcommand\SetHyphenMap[1]{%
2671   \bbl@forlang\bbl@tempa{%
2672     \expandafter\bbl@stringdef
2673     \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
2674 <</Macros local to BabelCommands>>
```

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

```
2675 \newcommand\BabelLower[2]{% one to one.
2676   \ifnum\lccode#1=#2\else
2677     \babel@savevariable{\lccode#1}%
2678     \lccode#1=#2\relax
2679   \fi}
2680 \newcommand\BabelLowerMM[4]{% many-to-many
2681   \@tempcnta=#1\relax
2682   \@tempcntb=#4\relax
2683   \def\bbl@tempa{%
2684     \ifnum\@tempcnta>#2\else
2685       \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
2686       \advance\@tempcnta#3\relax
2687       \advance\@tempcntb#3\relax
2688       \expandafter\bbl@tempa
2689     \fi}%
2690   \bbl@tempa}
2691 \newcommand\BabelLowerMO[4]{% many-to-one
2692   \@tempcnta=#1\relax
2693   \def\bbl@tempa{%
2694     \ifnum\@tempcnta>#2\else
2695       \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
2696       \advance\@tempcnta#3
2697       \expandafter\bbl@tempa
2698     \fi}%
2699   \bbl@tempa}
```

The following package options control the behavior of hyphenation mapping.

```
2700 <<*More package options>> ≡
2701 \DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\z@}
2702 \DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap\@ne}
2703 \DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\tw@}
2704 \DeclareOption{hyphenmap=other}{\chardef\bbl@opt@hyphenmap\thr@}
2705 \DeclareOption{hyphenmap=other*}{\chardef\bbl@opt@hyphenmap4\relax}
```



2706 <</More package options>>

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

```
2707 \AtEndOfPackage{%
2708   \ifx\bbbl@opt@hyphenmap\undefined
2709     \bbbl@xin@{,}{\bbbl@language@opts}%
2710     \chardef\bbbl@opt@hyphenmap\ifin@4\else\@ne\fi
2711   \fi}
```

This sections ends with a general tool for resetting the caption names with a unique interface. With the old way, which mixes the switcher and the string, we convert it to the new one, which separates these two steps.

```
2712 \newcommand\setlocalecaption{% TODO. Catch typos. What about ensure?
2713   \@ifstar\bbbl@setcaption@s\bbbl@setcaption@x}
2714 \def\bbbl@setcaption@x#1#2#3{% language caption-name string
2715   \bbbl@trim@def\bbbl@tempa{#2}%
2716   \bbbl@xin@{.template}{\bbbl@tempa}%
2717   \ifin@
2718     \bbbl@ini@captions@template{#3}{#1}%
2719   \else
2720     \edef\bbbl@tempd{%
2721       \expandafter\expandafter\expandafter
2722       \strip@prefix\expandafter\meaning\csname captions#1\endcsname}%
2723     \bbbl@xin@
2724       {\expandafter\string\csname #2name\endcsname}%
2725       {\bbbl@tempd}%
2726     \ifin@ % Renew caption
2727       \bbbl@xin@{\string\bbbl@scset}{\bbbl@tempd}%
2728     \ifin@
2729       \bbbl@exp{%
2730         \\bbbl@ifsamestring{\bbbl@tempa}{\language}%
2731         {\bbbl@scset\<#2name>\<#1#2name>}%
2732         {}}%
2733       \else % Old way converts to new way
2734         \bbbl@ifunset{#1#2name}%
2735         {\bbbl@exp{%
2736           \\bbbl@add\<captions#1>\def\<#2name>\<#1#2name>}}%
2737           \\bbbl@ifsamestring{\bbbl@tempa}{\language}%
2738           {\def\<#2name>\<#1#2name>}}%
2739           {}}%
2740         {}%
2741       \fi
2742     \else
2743       \bbbl@xin@{\string\bbbl@scset}{\bbbl@tempd}% New
2744       \ifin@ % New way
2745         \bbbl@exp{%
2746           \\bbbl@add\<captions#1>\bbbl@scset\<#2name>\<#1#2name>}%
2747           \\bbbl@ifsamestring{\bbbl@tempa}{\language}%
2748           {\bbbl@scset\<#2name>\<#1#2name>}%
2749           {}}%
2750         \else % Old way, but defined in the new way
2751         \bbbl@exp{%
2752           \\bbbl@add\<captions#1>\def\<#2name>\<#1#2name>}}%
2753           \\bbbl@ifsamestring{\bbbl@tempa}{\language}%
2754           {\def\<#2name>\<#1#2name>}}%
2755           {}}%
2756       \fi%
2757     \fi
2758     \@namedef{#1#2name}{#3}%
```

```

2759 \toks@\expandafter{\bbl@captionslist}%
2760 \bbl@exp{\in@{<#2name>}{\the\toks@}}%
2761 \ifin@else
2762 \bbl@exp{\bbl@add\bbl@captionslist{<#2name>}}%
2763 \bbl@tglobal\bbl@captionslist
2764 \fi
2765 \fi}
2766 % \def\bbl@setcaption@s#1#2#3{} % TODO. Not yet implemented

```

## 9.11 Macros common to a number of languages

`\set@low@box` The following macro is used to lower quotes to the same level as the comma. It prepares its argument in box register 0.

```

2767 \bbl@trace{Macros related to glyphs}
2768 \def\set@low@box#1{\setbox\tw@hbox{,}\setbox\z@hbox{#1}%
2769 \dimen\z@ht\z@ \advance\dimen\z@ -\ht\tw@%
2770 \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.

```

2771 \def\save@sf@q#1{\leavevmode
2772 \begingroup
2773 \edef\SF{\spacefactor\the\spacefactor}#1\SF
2774 \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.

```

2775 \ProvideTextCommand{\quotedblbase}{OT1}{%
2776 \save@sf@q{\set@low@box{\textquotedblright\}}%
2777 \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.

```

2778 \ProvideTextCommandDefault{\quotedblbase}{%
2779 \UseTextSymbol{OT1}{\quotedblbase}}

```

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

```

2780 \ProvideTextCommand{\quotesinglbase}{OT1}{%
2781 \save@sf@q{\set@low@box{\textquoteright\}}%
2782 \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.

```

2783 \ProvideTextCommandDefault{\quotesinglbase}{%
2784 \UseTextSymbol{OT1}{\quotesinglbase}}

```

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

```

2785 \ProvideTextCommand{\guillemetleft}{OT1}{%
2786 \ifmmode
2787 \ll
2788 \else

```

```

2789 \save@sf@q{\nobreak
2790 \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2791 \fi}
2792 \ProvideTextCommand{\guillemetright}{OT1}{%
2793 \ifmmode
2794 \gg
2795 \else
2796 \save@sf@q{\nobreak
2797 \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2798 \fi}
2799 \ProvideTextCommand{\guillemotleft}{OT1}{%
2800 \ifmmode
2801 \ll
2802 \else
2803 \save@sf@q{\nobreak
2804 \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2805 \fi}
2806 \ProvideTextCommand{\guillemotright}{OT1}{%
2807 \ifmmode
2808 \gg
2809 \else
2810 \save@sf@q{\nobreak
2811 \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2812 \fi}

```

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

```

2813 \ProvideTextCommandDefault{\guillemetleft}{%
2814 \UseTextSymbol{OT1}{\guillemetleft}}
2815 \ProvideTextCommandDefault{\guillemetright}{%
2816 \UseTextSymbol{OT1}{\guillemetright}}
2817 \ProvideTextCommandDefault{\guillemotleft}{%
2818 \UseTextSymbol{OT1}{\guillemotleft}}
2819 \ProvideTextCommandDefault{\guillemotright}{%
2820 \UseTextSymbol{OT1}{\guillemotright}}

```

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

```

2821 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2822 \ifmmode
2823 <%
2824 \else
2825 \save@sf@q{\nobreak
2826 \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
2827 \fi}
2828 \ProvideTextCommand{\guilsinglright}{OT1}{%
2829 \ifmmode
2830 >%
2831 \else
2832 \save@sf@q{\nobreak
2833 \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
2834 \fi}

```

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

```

2835 \ProvideTextCommandDefault{\guilsinglleft}{%
2836 \UseTextSymbol{OT1}{\guilsinglleft}}
2837 \ProvideTextCommandDefault{\guilsinglright}{%
2838 \UseTextSymbol{OT1}{\guilsinglright}}

```

### 9.12.2 Letters

`\ij` The dutch language uses the letter 'ij'. It is available in T1 encoded fonts, but not in the OT1 encoded  
`\IJ` fonts. Therefore we fake it for the OT1 encoding.

```
2839 \DeclareTextCommand{\ij}{OT1}{%
2840   i\kern-0.02em\bbl@allowhyphens j}
2841 \DeclareTextCommand{\IJ}{OT1}{%
2842   I\kern-0.02em\bbl@allowhyphens J}
2843 \DeclareTextCommand{\ij}{T1}{\char188}
2844 \DeclareTextCommand{\IJ}{T1}{\char156}
```

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

```
2845 \ProvideTextCommandDefault{\ij}{%
2846   \UseTextSymbol{OT1}{\ij}}
2847 \ProvideTextCommandDefault{\IJ}{%
2848   \UseTextSymbol{OT1}{\IJ}}
```

`\dj` The croatian language needs the letters `\dj` and `\DJ`; they are available in the T1 encoding, but not in  
`\DJ` the OT1 encoding by default.

Some code to construct these glyphs for the OT1 encoding was made available to me by Stipčević Mario, (stipcevic@olimp.irb.hr).

```
2849 \def\crrtic@{\hrule height0.1ex width0.3em}
2850 \def\crttic@{\hrule height0.1ex width0.33em}
2851 \def\ddj@{%
2852   \setbox0\hbox{d}\dimen@=\ht0
2853   \advance\dimen@1ex
2854   \dimen@.45\dimen@
2855   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2856   \advance\dimen@ii.5ex
2857   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2858 \def\DDJ@{%
2859   \setbox0\hbox{D}\dimen@=.55\ht0
2860   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2861   \advance\dimen@ii.15ex % correction for the dash position
2862   \advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
2863   \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
2864   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
2865 %
2866 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2867 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}
```

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

```
2868 \ProvideTextCommandDefault{\dj}{%
2869   \UseTextSymbol{OT1}{\dj}}
2870 \ProvideTextCommandDefault{\DJ}{%
2871   \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.

```
2872 \DeclareTextCommand{\SS}{OT1}{SS}
2873 \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
2874 \ProvideTextCommandDefault{\glq}{%
2875   \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}

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

2876 \ProvideTextCommand{\grq}{T1}{%
2877   \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}
2878 \ProvideTextCommand{\grq}{TU}{%
2879   \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
2880 \ProvideTextCommand{\grq}{OT1}{%
2881   \save@sf@q{\kern-.0125em
2882     \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%
2883     \kern.07em\relax}}
2884 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}\grq}
```

`\glqq` The ‘german’ double quotes.

```
\grqq
2885 \ProvideTextCommandDefault{\glqq}{%
2886   \textormath{\quotedblbase}{\mbox{\quotedblbase}}}

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

2887 \ProvideTextCommand{\grqq}{T1}{%
2888   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2889 \ProvideTextCommand{\grqq}{TU}{%
2890   \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2891 \ProvideTextCommand{\grqq}{OT1}{%
2892   \save@sf@q{\kern-.07em
2893     \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
2894     \kern.07em\relax}}
2895 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}
```

`\flq` The ‘french’ single guillemets.

```
\frq
2896 \ProvideTextCommandDefault{\flq}{%
2897   \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
2898 \ProvideTextCommandDefault{\frq}{%
2899   \textormath{\guilsinglright}{\mbox{\guilsinglright}}}
```

`\flqq` The ‘french’ double guillemets.

```
\frqq
2900 \ProvideTextCommandDefault{\flqq}{%
2901   \textormath{\guillemetleft}{\mbox{\guillemetleft}}}
2902 \ProvideTextCommandDefault{\frqq}{%
2903   \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 positioning, the  
`\umlautlow` default will be `\umlauthigh` (the normal positioning).

```
2904 \def\umlauthigh{%
2905   \def\bbl@umlauta##1{\leavevmode\bgroup%
2906     \expandafter\accent\csname\fontencoding dpos\endcsname
2907     ##1\bbl@allowhyphens\egroup}%
2908   \let\bbl@umlaute\bbl@umlauta}
2909 \def\umlautlow{%
2910   \def\bbl@umlauta{\protect\lower@umlaut}}
```

```

2911 \def\umlaute\lower{%
2912   \def\bbl@umlaute{\protect\lower@umlaut}}
2913 \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.

```

2914 \expandafter\ifx\csname U@D\endcsname\relax
2915   \csname newdimen\endcsname\U@D
2916 \fi

```

The following code fools T<sub>E</sub>X'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.

```

2917 \def\lower@umlaut#1{%
2918   \leavevmode\bggroup
2919     \U@D 1ex%
2920     {\setbox\z@\hbox{%
2921       \expandafter\char\csname\fontencoding dqpos\endcsname}%
2922       \dimen@ -.45ex\advance\dimen@\ht\z@
2923       \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2924     \expandafter\accent\csname\fontencoding dqpos\endcsname
2925     \fontdimen5\font\U@D #1%
2926   \egroup}

```

For all vowels we declare `\` to be a composite command which uses `\bbl@umlaut` 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@umlaut` and/or `\bbl@umlaute` for a language in the corresponding `ldf` (using the `babel` switching mechanism, of course).

```

2927 \AtBeginDocument{%
2928   \DeclareTextCompositeCommand{\}{OT1}{a}{\bbl@umlaut{a}}%
2929   \DeclareTextCompositeCommand{\}{OT1}{e}{\bbl@umlaute{e}}%
2930   \DeclareTextCompositeCommand{\}{OT1}{i}{\bbl@umlaute{i}}%
2931   \DeclareTextCompositeCommand{\}{OT1}{\i}{\bbl@umlaute{i}}%
2932   \DeclareTextCompositeCommand{\}{OT1}{o}{\bbl@umlaut{o}}%
2933   \DeclareTextCompositeCommand{\}{OT1}{u}{\bbl@umlaut{u}}%
2934   \DeclareTextCompositeCommand{\}{OT1}{A}{\bbl@umlaut{A}}%
2935   \DeclareTextCompositeCommand{\}{OT1}{E}{\bbl@umlaute{E}}%
2936   \DeclareTextCompositeCommand{\}{OT1}{I}{\bbl@umlaute{I}}%
2937   \DeclareTextCompositeCommand{\}{OT1}{O}{\bbl@umlaut{O}}%
2938   \DeclareTextCompositeCommand{\}{OT1}{U}{\bbl@umlaut{U}}%

```

Finally, make sure the default hyphenrules are defined (even if empty). For internal use, another empty `\language` is defined. Currently used in Amharic.

```

2939 \ifx\l@english\undefined
2940   \chardef\l@english\z@
2941 \fi
2942 % The following is used to cancel rules in ini files (see Amharic).
2943 \ifx\l@babelnohyphens\undefined
2944   \newlanguage\l@babelnohyphens
2945 \fi

```

## 9.13 Layout

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

```
2946 \bbl@trace{Bidi layout}
2947 \providecommand\IfBabelLayout[3]{#3}%
2948 \newcommand\BabelPatchSection[1]{%
2949   \@ifundefined{#1}{}{%
2950     \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
2951     \@namedef{#1}{%
2952       \ifstar{\bbl@presec@s{#1}}%
2953       {\@dblarg{\bbl@presec@x{#1}}}}}%
2954 \def\bbl@presec@x#1[#2]#3{%
2955   \bbl@exp{%
2956     \\\select@language@x{\bbl@main@language}%
2957     \\\bbl@cs{sspre@#1}%
2958     \\\bbl@cs{ss@#1}%
2959     [\\foreignlanguage{\language}{\unexpanded{#2}}]%
2960     {\\foreignlanguage{\language}{\unexpanded{#3}}}%
2961     \\\select@language@x{\language}}}%
2962 \def\bbl@presec@s#1#2{%
2963   \bbl@exp{%
2964     \\\select@language@x{\bbl@main@language}%
2965     \\\bbl@cs{sspre@#1}%
2966     \\\bbl@cs{ss@#1}*%
2967     {\\foreignlanguage{\language}{\unexpanded{#2}}}%
2968     \\\select@language@x{\language}}}%
2969 \IfBabelLayout{sectioning}%
2970   {\BabelPatchSection{part}%
2971    \BabelPatchSection{chapter}%
2972    \BabelPatchSection{section}%
2973    \BabelPatchSection{subsection}%
2974    \BabelPatchSection{subsubsection}%
2975    \BabelPatchSection{paragraph}%
2976    \BabelPatchSection{subparagraph}%
2977    \def\babel@toc#1{%
2978      \select@language@x{\bbl@main@language}}}%
2979 \IfBabelLayout{captions}%
2980   {\BabelPatchSection{caption}}}
```

## 9.14 Load engine specific macros

```
2981 \bbl@trace{Input engine specific macros}
2982 \ifcase\bbl@engine
2983   \input txtbabel.def
2984 \or
2985   \input luababel.def
2986 \or
2987   \input xebabel.def
2988 \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 previously loaded ldf files.

```
2989 \bbl@trace{Creating languages and reading ini files}
2990 \newcommand\babelprovide[2][{}]{%
2991   \let\bbl@savelangname\language
2992   \edef\bbl@savelocaleid{\the\localeid}%
```

```

2993 % Set name and locale id
2994 \edef\languagename{#2}%
2995 % \global\@namedef{bbl@lcname@#2}{#2}%
2996 \bbl@id@assign
2997 \let\bbl@KVP@captions\@nil
2998 \let\bbl@KVP@date\@nil
2999 \let\bbl@KVP@import\@nil
3000 \let\bbl@KVP@main\@nil
3001 \let\bbl@KVP@script\@nil
3002 \let\bbl@KVP@language\@nil
3003 \let\bbl@KVP@hyphenrules\@nil
3004 \let\bbl@KVP@mapfont\@nil
3005 \let\bbl@KVP@maparabic\@nil
3006 \let\bbl@KVP@mapdigits\@nil
3007 \let\bbl@KVP@intraspace\@nil
3008 \let\bbl@KVP@intrapenalty\@nil
3009 \let\bbl@KVP@onchar\@nil
3010 \let\bbl@KVP@transforms\@nil
3011 \global\let\bbl@release@transforms\@empty
3012 \let\bbl@KVP@alph\@nil
3013 \let\bbl@KVP@Alph\@nil
3014 \let\bbl@KVP@labels\@nil
3015 \bbl@csarg\let{KVP@labels*}\@nil
3016 \global\let\bbl@inidata\@empty
3017 \bbl@forkv{#1}{% TODO - error handling
3018   \in@{/}{##1}%
3019   \ifin@
3020     \bbl@renewinikey##1\@@{##2}%
3021   \else
3022     \bbl@csarg\def{KVP@##1}{##2}%
3023   \fi}%
3024 % == init ==
3025 \ifx\bbl@screset\@undefined
3026   \bbl@ldfinit
3027 \fi
3028 % ==
3029 \let\bbl@lbkflag\relax % \@empty = do setup linebreak
3030 \bbl@ifunset{date#2}%
3031   {\let\bbl@lbkflag\@empty}% new
3032   {\ifx\bbl@KVP@hyphenrules\@nil\else
3033     \let\bbl@lbkflag\@empty
3034   \fi
3035   \ifx\bbl@KVP@import\@nil\else
3036     \let\bbl@lbkflag\@empty
3037   \fi}%
3038 % == import, captions ==
3039 \ifx\bbl@KVP@import\@nil\else
3040   \bbl@exp{\@bbl@ifblank{\bbl@KVP@import}}%
3041   {\ifx\bbl@initload\relax
3042     \begingroup
3043       \def\BabelBeforeIni##1##2{\gdef\bbl@KVP@import{##1}\endinput}%
3044       \bbl@input@texini{#2}%
3045     \endgroup
3046   \else
3047     \xdef\bbl@KVP@import{\bbl@initload}%
3048   \fi}%
3049   {}%
3050 \fi
3051 \ifx\bbl@KVP@captions\@nil

```



```

3052 \let\bbl@KVP@captions\bbl@KVP@import
3053 \fi
3054 % ==
3055 \ifx\bbl@KVP@transforms\@nil\else
3056 \bbl@replace\bbl@KVP@transforms{ }{,}%
3057 \fi
3058 % Load ini
3059 \bbl@ifunset{date#2}%
3060 {\bbl@provide@new{#2}}%
3061 {\bbl@ifblank{#1}%
3062 {}% With \bbl@load@basic below
3063 {\bbl@provide@renew{#2}}}%
3064 % Post tasks
3065 % -----
3066 % == ensure captions ==
3067 \ifx\bbl@KVP@captions\@nil\else
3068 \bbl@ifunset{bbl@extracaps#2}%
3069 {\bbl@exp{\bbl@babelensure[exclude=\today]{#2}}}%
3070 {\toks@ \expandafter \expandafter \expandafter
3071 {\csname bbl@extracaps#2\endcsname}%
3072 \bbl@exp{\bbl@babelensure[exclude=\today,include=\the\toks@]{#2}}%
3073 \bbl@ifunset{bbl@ensure@\language}%
3074 {\bbl@exp{%
3075 \\\DeclareRobustCommand\<bbl@ensure@\language>[1]{%
3076 \\\foreignlanguage{\language}%
3077 {###1}}}%
3078 }%
3079 \bbl@exp{%
3080 \\\bbl@tglobal\<bbl@ensure@\language>%
3081 \\\bbl@tglobal\<bbl@ensure@\language\space>%
3082 \fi
3083 % ==
3084 % At this point all parameters are defined if 'import'. Now we
3085 % execute some code depending on them. But what about if nothing was
3086 % imported? We just set the basic parameters, but still loading the
3087 % whole ini file.
3088 \bbl@load@basic{#2}%
3089 % == script, language ==
3090 % Override the values from ini or defines them
3091 \ifx\bbl@KVP@script\@nil\else
3092 \bbl@csarg\edef{sname#2}{\bbl@KVP@script}%
3093 \fi
3094 \ifx\bbl@KVP@language\@nil\else
3095 \bbl@csarg\edef{lname#2}{\bbl@KVP@language}%
3096 \fi
3097 % == onchar ==
3098 \ifx\bbl@KVP@onchar\@nil\else
3099 \bbl@luahyphenate
3100 \directlua{
3101 if Babel.locale_mapped == nil then
3102 Babel.locale_mapped = true
3103 Babel.linebreaking.add_before(Babel.locale_map)
3104 Babel.loc_to_scr = {}
3105 Babel.chr_to_loc = Babel.chr_to_loc or {}
3106 end}%
3107 \bbl@xin@{ ids }{ \bbl@KVP@onchar\space}%
3108 \ifin@
3109 \ifx\bbl@starthyphens\undefined % Needed if no explicit selection
3110 \AddBabelHook{babel-onchar}{beforestart}{\bbl@starthyphens}%

```

```

3111 \fi
3112 \bbl@exp{\bbl@add\bbl@starthyphens
3113 {\bbl@patterns@lua{\language}}}%
3114 % TODO - error/warning if no script
3115 \directlua{
3116   if Babel.script_blocks['\bbl@cl{sbc}'] then
3117     Babel.loc_to_scr[\the\localeid] =
3118       Babel.script_blocks['\bbl@cl{sbc}']
3119     Babel.locale_props[\the\localeid].lc = \the\localeid\space
3120     Babel.locale_props[\the\localeid].lg = \the\nameuse{1@\language}\space
3121   end
3122 }%
3123 \fi
3124 \bbl@xin@{ fonts }{ \bbl@KVP@onchar\space}%
3125 \ifin@
3126 \bbl@ifunset{bbl@lsys@\language}{\bbl@provide@lsys{\language}}}%
3127 \bbl@ifunset{bbl@wdir@\language}{\bbl@provide@dirs{\language}}}%
3128 \directlua{
3129   if Babel.script_blocks['\bbl@cl{sbc}'] then
3130     Babel.loc_to_scr[\the\localeid] =
3131       Babel.script_blocks['\bbl@cl{sbc}']
3132   end}%
3133 \ifx\bbl@mapselect\undefined % TODO. almost the same as mapfont
3134 \AtBeginDocument{%
3135   \expandafter\bbl@add\csname selectfont \endcsname{\bbl@mapselect}}%
3136   {\selectfont}}%
3137 \def\bbl@mapselect{%
3138   \let\bbl@mapselect\relax
3139   \edef\bbl@prefontid{\fontid\font}}%
3140 \def\bbl@mapdir##1{%
3141   {\def\language{##1}%
3142     \let\bbl@ifrestoring\@firstoftwo % To avoid font warning
3143     \bbl@switchfont
3144     \directlua{
3145       Babel.locale_props[\the\csname bbl@id@##1\endcsname]
3146         [\bbl@prefontid] = \fontid\font\space}}}%
3147 \fi
3148 \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\language}}}%
3149 \fi
3150 % TODO - catch non-valid values
3151 \fi
3152 % == mapfont ==
3153 % For bidi texts, to switch the font based on direction
3154 \ifx\bbl@KVP@mapfont\@nil\else
3155 \bbl@ifsamestring{\bbl@KVP@mapfont}{direction}}%
3156 {\bbl@error{Option '\bbl@KVP@mapfont' unknown for\
3157   mapfont. Use 'direction'.%
3158   {See the manual for details.}}}%
3159 \bbl@ifunset{bbl@lsys@\language}{\bbl@provide@lsys{\language}}}%
3160 \bbl@ifunset{bbl@wdir@\language}{\bbl@provide@dirs{\language}}}%
3161 \ifx\bbl@mapselect\undefined % TODO. See onchar
3162 \AtBeginDocument{%
3163   \expandafter\bbl@add\csname selectfont \endcsname{\bbl@mapselect}}%
3164   {\selectfont}}%
3165 \def\bbl@mapselect{%
3166   \let\bbl@mapselect\relax
3167   \edef\bbl@prefontid{\fontid\font}}%
3168 \def\bbl@mapdir##1{%
3169   {\def\language{##1}%

```

```

3170         \let\bbl@ifrestoring\@firstoftwo % avoid font warning
3171         \bbl@switchfont
3172         \directlua{Babel.fontmap
3173             [\the\csname bbl@wdir@##1\endcsname]%
3174             [\bbl@prefontid]=\fontid\font}}}%
3175     \fi
3176     \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\language}}}%
3177 \fi
3178 % == Line breaking: intraspace, intrapenalty ==
3179 % For CJK, East Asian, Southeast Asian, if interspace in ini
3180 \ifx\bbl@KVP@intraspace\@nil\else % We can override the ini or set
3181     \bbl@csarg\edef{intsp@#2}{\bbl@KVP@intraspace}%
3182 \fi
3183 \bbl@provide@intraspace
3184 % == Line breaking: hyphenate.other.locale/.script==
3185 \ifx\bbl@lbkflag\@empty
3186     \bbl@ifunset{bbl@hyotl@language}{}%
3187     {\bbl@csarg\bbl@replace{hyotl@language}{ }{,}%
3188     \bbl@startcommands*{language}{%
3189     \bbl@csarg\bbl@foreach{hyotl@language}{%
3190     \ifcase\bbl@engine
3191     \ifnum##1<257
3192     \SetHyphenMap{\BabelLower{##1}{##1}}%
3193     \fi
3194     \else
3195     \SetHyphenMap{\BabelLower{##1}{##1}}%
3196     \fi}%
3197     \bbl@endcommands}%
3198 \bbl@ifunset{bbl@hyots@language}{}%
3199 {\bbl@csarg\bbl@replace{hyots@language}{ }{,}%
3200 \bbl@csarg\bbl@foreach{hyots@language}{%
3201 \ifcase\bbl@engine
3202 \ifnum##1<257
3203 \global\lccode##1=##1\relax
3204 \fi
3205 \else
3206 \global\lccode##1=##1\relax
3207 \fi}}%
3208 \fi
3209 % == Counters: maparabic ==
3210 % Native digits, if provided in ini (TeX level, xe and lua)
3211 \ifcase\bbl@engine\else
3212     \bbl@ifunset{bbl@dgnat@language}{}%
3213     {\expandafter\ifx\csname bbl@dgnat@language\endcsname\@empty\else
3214     \expandafter\expandafter\expandafter
3215     \bbl@setdigits\csname bbl@dgnat@language\endcsname
3216     \ifx\bbl@KVP@maparabic\@nil\else
3217     \ifx\bbl@latinarabic\@undefined
3218     \expandafter\let\expandafter\@arabic
3219     \csname bbl@counter@language\endcsname
3220     \else % ie, if layout=counters, which redefines \@arabic
3221     \expandafter\let\expandafter\bbl@latinarabic
3222     \csname bbl@counter@language\endcsname
3223     \fi
3224     \fi
3225     \fi}%
3226 \fi
3227 % == Counters: mapdigits ==
3228 % Native digits (lua level).

```

```

3229 \ifodd\bbl@engine
3230 \ifx\bbl@KVP@mapdigits\@nil\else
3231 \bbl@ifunset{bbl@dgnat\language\name}{}%
3232 {\RequirePackage{luatexbase}%
3233 \bbl@activate@preotf
3234 \directlua{
3235     Babel = Babel or {} %% -> presets in luababel
3236     Babel.digits_mapped = true
3237     Babel.digits = Babel.digits or {}
3238     Babel.digits[\the\localeid] =
3239     table.pack(string.utfvalue('\bbl@cl{dgnat}'))
3240     if not Babel.numbers then
3241     function Babel.numbers(head)
3242         local LOCALE = luatexbase.registernumber'bbl@attr@locale'
3243         local GLYPH = node.id'glyph'
3244         local inmath = false
3245         for item in node.traverse(head) do
3246             if not inmath and item.id == GLYPH then
3247                 local temp = node.get_attribute(item, LOCALE)
3248                 if Babel.digits[temp] then
3249                     local chr = item.char
3250                     if chr > 47 and chr < 58 then
3251                         item.char = Babel.digits[temp][chr-47]
3252                     end
3253                 end
3254             elseif item.id == node.id'math' then
3255                 inmath = (item.subtype == 0)
3256             end
3257         end
3258         return head
3259     end
3260     end
3261 }}%
3262 \fi
3263 \fi
3264 % == Counters: alph, Alph ==
3265 % What if extras<lang> contains a \babel@save\@alph? It won't be
3266 % restored correctly when exiting the language, so we ignore
3267 % this change with the \bbl@alph@saved trick.
3268 \ifx\bbl@KVP@alph\@nil\else
3269 \toks@\expandafter\expandafter\expandafter{%
3270 \csname extras\language\endcsname}%
3271 \bbl@exp{%
3272 \def\<extras\language>{%
3273 \let\\bbl@alph@saved\\@alph
3274 \the\toks@
3275 \let\\@alph\\bbl@alph@saved
3276 \\babel@save\\@alph
3277 \let\\@alph<bbl@cntr\bbl@KVP@alph @\language>}}%
3278 \fi
3279 \ifx\bbl@KVP@Alph\@nil\else
3280 \toks@\expandafter\expandafter\expandafter{%
3281 \csname extras\language\endcsname}%
3282 \bbl@exp{%
3283 \def\<extras\language>{%
3284 \let\\bbl@Alph@saved\\@Alph
3285 \the\toks@
3286 \let\\@Alph\\bbl@Alph@saved
3287 \\babel@save\\@Alph

```

```

3288 \let\\@Alph\<bbl@cntr@\bbl@KVP@Alph @\languagename>}}%
3289 \fi
3290 % == require.babel in ini ==
3291 % To load or reload the babel-*.tex, if require.babel in ini
3292 \ifx\bbl@beforestart\relax\else % But not in doc aux or body
3293 \bbl@ifunset{bbl@rtex@\languagename}{}%
3294 {\expandafter\ifx\csname bbl@rtex@\languagename\endcsname\empty\else
3295 \let\BabelBeforeIni\@gobbletwo
3296 \chardef\atcatcode=\catcode`\@
3297 \catcode`\@=11\relax
3298 \bbl@input@texini{\bbl@cs{rtex@\languagename}}}%
3299 \catcode`\@=\atcatcode
3300 \let\atcatcode\relax
3301 \fi}%
3302 \fi
3303 % == Release saved transforms ==
3304 \bbl@release@transforms\relax % \relax closes the last item.
3305 % == main ==
3306 \ifx\bbl@KVP@main\@nil % Restore only if not 'main'
3307 \let\languagename\bbl@savelangname
3308 \chardef\localeid\bbl@savelocaleid\relax
3309 \fi}

```

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

```

3310 \def\bbl@provide@new#1{%
3311 \namedef{date#1}{}}% marks lang exists - required by \StartBabelCommands
3312 \namedef{extras#1}{}%
3313 \namedef{noextras#1}{}%
3314 \bbl@startcommands*{#1}{captions}%
3315 \ifx\bbl@KVP@captions\@nil % and also if import, implicit
3316 \def\bbl@tempb##1{% elt for \bbl@captionslist
3317 \ifx##1\empty\else
3318 \bbl@exp{%
3319 \SetString\##1{%
3320 \bbl@nocaption{\bbl@stripslash##1}{#1\bbl@stripslash##1}}}%
3321 \expandafter\bbl@tempb
3322 \fi}%
3323 \expandafter\bbl@tempb\bbl@captionslist\empty
3324 \else
3325 \ifx\bbl@initoload\relax
3326 \bbl@read@ini{\bbl@KVP@captions}2% % Here letters cat = 11
3327 \else
3328 \bbl@read@ini{\bbl@initoload}2% % Same
3329 \fi
3330 \fi
3331 \StartBabelCommands*{#1}{date}%
3332 \ifx\bbl@KVP@import\@nil
3333 \bbl@exp{%
3334 \SetString\today{\bbl@nocaption{today}{#1today}}}%
3335 \else
3336 \bbl@savetoday
3337 \bbl@savedate
3338 \fi
3339 \bbl@endcommands
3340 \bbl@load@basic{#1}%
3341 % == hyphenmins == (only if new)
3342 \bbl@exp{%
3343 \gdef\<#1hyphenmins>{%
3344 {\bbl@ifunset{bbl@ifthm@#1}{2}{\bbl@cs{lfthm@#1}}}%

```

```

3345     {\bbl@ifunset{\bbl@rgthm@#1}{3}{\bbl@cs{rgthm@#1}}}}}%
3346 % == hyphenrules ==
3347 \bbl@provide@hyphens{#1}%
3348 % == frenchspacing == (only if new)
3349 \bbl@ifunset{\bbl@frspc@#1}{}%
3350 {\edef\bbl@tempa{\bbl@c1{frspc}}}%
3351 \edef\bbl@tempa{\expandafter\@car\bbl@tempa\@nil}%
3352 \if u\bbl@tempa % do nothing
3353 \else\if n\bbl@tempa % non french
3354 \expandafter\bbl@add\csname extras#1\endcsname{%
3355 \let\bbl@elt\bbl@fs@elt@i
3356 \bbl@fs@chars}%
3357 \else\if y\bbl@tempa % french
3358 \expandafter\bbl@add\csname extras#1\endcsname{%
3359 \let\bbl@elt\bbl@fs@elt@ii
3360 \bbl@fs@chars}%
3361 \fi\fi\fi}%
3362 %
3363 \ifx\bbl@KVP@main\@nil\else
3364 \expandafter\main@language\expandafter{#1}%
3365 \fi}
3366 % A couple of macros used above, to avoid hashes #####...
3367 \def\bbl@fs@elt@i#1#2#3{%
3368 \ifnum\sfcode`#1=#2\relax
3369 \babel@savevariable{\sfcode`#1}%
3370 \sfcode`#1=#3\relax
3371 \fi}%
3372 \def\bbl@fs@elt@ii#1#2#3{%
3373 \ifnum\sfcode`#1=#3\relax
3374 \babel@savevariable{\sfcode`#1}%
3375 \sfcode`#1=#2\relax
3376 \fi}%
3377 %
3378 \def\bbl@provide@renew#1{%
3379 \ifx\bbl@KVP@captions\@nil\else
3380 \StartBabelCommands*{#1}{captions}%
3381 \bbl@read@ini{\bbl@KVP@captions}2% % Here all letters cat = 11
3382 \EndBabelCommands
3383 \fi
3384 \ifx\bbl@KVP@import\@nil\else
3385 \StartBabelCommands*{#1}{date}%
3386 \bbl@savetoday
3387 \bbl@savestate
3388 \EndBabelCommands
3389 \fi
3390 % == hyphenrules ==
3391 \ifx\bbl@lbkflag\@empty
3392 \bbl@provide@hyphens{#1}%
3393 \fi}

```

Load the basic parameters (ids, typography, counters, and a few more), while captions and dates are left out. But it may happen some data has been loaded before automatically, so we first discard the saved values.

```

3394 \def\bbl@load@basic#1{%
3395 \bbl@ifunset{\bbl@inidata@\language}\@nil}%
3396 {\getlocaleproperty\bbl@tempa{\language}{identification/load.level}%
3397 \ifcase\bbl@tempa
3398 \bbl@csarg\let{lname@\language}\relax
3399 \fi}%

```

```

3400 \bbl@ifunset{bbl@lname@#1}%
3401 {\def\BabelBeforeIni##1##2{%
3402   \begingroup
3403     \let\bbl@ini@captions@aux\@gobbletwo
3404     \def\bbl@inidate ####1.####2.####3.####4\relax ####5####6}%
3405     \bbl@read@ini{##1}1%
3406     \ifx\bbl@initoload\relax\endinput\fi
3407   \endgroup}%
3408   \begingroup      % boxed, to avoid extra spaces:
3409   \ifx\bbl@initoload\relax
3410     \bbl@input@texini{##1}%
3411   \else
3412     \setbox\z@\hbox{\BabelBeforeIni{\bbl@initoload}}}%
3413   \fi
3414 \endgroup}%
3415 {}

```

The hyphenrules option is handled with an auxiliary macro.

```

3416 \def\bbl@provide@hyphens#1{%
3417   \let\bbl@tempa\relax
3418   \ifx\bbl@KVP@hyphenrules\@nil\else
3419     \bbl@replace\bbl@KVP@hyphenrules{ }{,}%
3420     \bbl@foreach\bbl@KVP@hyphenrules{%
3421       \ifx\bbl@tempa\relax      % if not yet found
3422         \bbl@ifsamestring{##1}{+}%
3423         {\bbl@exp{\addlanguage\<l@##1>}}}%
3424       {}%
3425       \bbl@ifunset{l@##1}%
3426       {}%
3427       {\bbl@exp{\let\bbl@tempa\<l@##1>}}}%
3428     \fi}%
3429 \fi
3430 \ifx\bbl@tempa\relax %          if no opt or no language in opt found
3431   \ifx\bbl@KVP@import\@nil
3432     \ifx\bbl@initoload\relax\else
3433       \bbl@exp{%
3434         \bbl@ifblank{\bbl@cs{hyphr@#1}}%
3435         {}%
3436         {\let\bbl@tempa\<l@bbl@cl{hyphr}>}}%
3437     \fi
3438   \else % if importing
3439     \bbl@exp{%
3440       \bbl@ifblank{\bbl@cs{hyphr@#1}}%
3441       {}%
3442       {\let\bbl@tempa\<l@bbl@cl{hyphr}>}}%
3443   \fi
3444 \fi
3445 \bbl@ifunset{bbl@tempa}%      ie, relax or undefined
3446 {\bbl@ifunset{l@#1}%          no hyphenrules found - fallback
3447   {\bbl@exp{\adddialect\<l@#1>\language}}%
3448   {}%
3449   {\bbl@exp{\adddialect\<l@#1>\bbl@tempa}}}% found in opt list or ini

```

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

```

3450 \def\bbl@input@texini#1{%
3451   \bbl@bsphack
3452   \bbl@exp{%
3453     \catcode`\\=14 \catcode`\\=0
3454     \catcode`\\={1 \catcode`\\}=2

```

```

3455 \lowercase{\InputIfFileExists{babel-#1.tex}{}}}%
3456 \catcode`\%%=\the\catcode`\%\relax
3457 \catcode`\%%=\the\catcode`\%\relax
3458 \catcode`\%=\the\catcode`\%\relax
3459 \catcode`\%=\the\catcode`\%\relax}%
3460 \bbl@esphack}

```

The following macros read and store ini files (but don't process them). For each line, there are 3 possible actions: ignore if starts with ;, switch section if starts with [, and store otherwise. There are used in the first step of \bbl@read@ini.

```

3461 \def\bbl@inline#1\bbl@inline{%
3462 \@ifnextchar[\bbl@inisect{\@ifnextchar;\bbl@iniskip\bbl@inistore}#1\@@}% ]
3463 \def\bbl@inisect[#1]#2\@@{\def\bbl@section{#1}}%
3464 \def\bbl@iniskip#1\@@{% if starts with ;
3465 \def\bbl@inistore#1=#2\@@{% full (default)
3466 \bbl@trim@def\bbl@tempa{#1}%
3467 \bbl@trim\toks@{#2}%
3468 \bbl@ifunset{bbl@KVP@\bbl@section/\bbl@tempa}%
3469 {\bbl@exp{%
3470 \\\g@addto@macro\\bbl@inidata{%
3471 \\\bbl@elt{\bbl@section}{\bbl@tempa}{\the\toks@}}}%
3472 }}}%
3473 \def\bbl@inistore@min#1=#2\@@{% minimal (maybe set in \bbl@read@ini)
3474 \bbl@trim@def\bbl@tempa{#1}%
3475 \bbl@trim\toks@{#2}%
3476 \bbl@xin@{.identification.}{.\bbl@section.}%
3477 \ifin@
3478 \bbl@exp{\\g@addto@macro\\bbl@inidata{%
3479 \\\bbl@elt{identification}{\bbl@tempa}{\the\toks@}}}%
3480 \fi}%

```

Now, the 'main loop', which **\*\*must be executed inside a group\*\***. At this point, \bbl@inidata may contain data declared in \babelprovide, with 'slashed' keys. There are 3 steps: first read the ini file and store it; then traverse the stored values, and process some groups if required (date, captions, labels, counters); finally, 'export' some values by defining global macros (identification, typography, characters, numbers). The second argument is 0 when called to read the minimal data for fonts; with \babelprovide it's either 1 or 2.

```

3481 \ifx\bbl@readstream\@undefined
3482 \csname newread\endcsname\bbl@readstream
3483 \fi
3484 \def\bbl@read@ini#1#2{%
3485 \openin\bbl@readstream=babel-#1.ini
3486 \ifeof\bbl@readstream
3487 \bbl@error
3488 {There is no ini file for the requested language\%
3489 (#1). Perhaps you misspelled it or your installation\%
3490 is not complete.}%
3491 {Fix the name or reinstall babel.}%
3492 \else
3493 % Store ini data in \bbl@inidata
3494 \catcode`\[=12 \catcode`\]=12 \catcode`\==12 \catcode`\&=12
3495 \catcode`\|=12 \catcode`\|=12 \catcode`\%=14 \catcode`\-=12
3496 \bbl@info{Importing
3497 \ifcase#2font and identification \or basic \fi
3498 data for \language\%
3499 from babel-#1.ini. Reported}%
3500 \ifnum#2=\z@
3501 \global\let\bbl@inidata\@empty
3502 \let\bbl@inistore\bbl@inistore@min % Remember it's local

```



```

3503 \fi
3504 \def\bbl@section{identification}%
3505 \bbl@exp{\bbl@inistore tag.ini=#1\\@@}%
3506 \bbl@inistore load.level=#2\\@@
3507 \loop
3508 \if T\ifeof\bbl@readstream F\fi T\relax % Trick, because inside \loop
3509 \endlinechar@m@ne
3510 \read\bbl@readstream to \bbl@line
3511 \endlinechar`^^M
3512 \ifx\bbl@line\@empty\else
3513 \expandafter\bbl@iniline\bbl@line\bbl@iniline
3514 \fi
3515 \repeat
3516 % Process stored data
3517 \bbl@csarg\xdef{lini@languagename}{#1}%
3518 \let\bbl@savestrings\@empty
3519 \let\bbl@savetoday\@empty
3520 \let\bbl@savedate\@empty
3521 \def\bbl@elt##1##2##3{%
3522 \def\bbl@section{##1}%
3523 \in@{=date.}{=##1}% Find a better place
3524 \ifin@
3525 \bbl@ini@calendar{##1}%
3526 \fi
3527 \global\bbl@csarg\let{bbl@KVP@##1/##2}\relax
3528 \bbl@ifunset{bbl@inikv@##1}{}%
3529 { \csname bbl@inikv@##1\endcsname{##2}{##3}}}%
3530 \bbl@inidata
3531 % 'Export' data
3532 \bbl@ini@exports{#2}%
3533 \global\bbl@csarg\let{inidata@languagename}\bbl@inidata
3534 \global\let\bbl@inidata\@empty
3535 \bbl@exp{\bbl@add@list\bbl@ini@loaded{languagename}}%
3536 \bbl@tglobal\bbl@ini@loaded
3537 \fi}

```

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

```

3538 \def\bbl@ini@calendar#1{%
3539 \lowercase{\def\bbl@tempa{=#1=}}%
3540 \bbl@replace\bbl@tempa{=date.gregorian}{}%
3541 \bbl@replace\bbl@tempa{=date.}{}%
3542 \in@{.licr=}{#1=}%
3543 \ifin@
3544 \ifcase\bbl@engine
3545 \bbl@replace\bbl@tempa{.licr=}{}%
3546 \else
3547 \let\bbl@tempa\relax
3548 \fi
3549 \fi
3550 \ifx\bbl@tempa\relax\else
3551 \bbl@replace\bbl@tempa{=}{}%
3552 \bbl@exp{%
3553 \def<bbl@inikv@#1>####1####2{%
3554 \bbl@inidate####1...\relax{####2}{\bbl@tempa}}}%
3555 \fi}

```

A key with a slash in \babelprovide replaces the value in the ini file (which is ignored altogether). The mechanism is simple (but suboptimal): add the data to the ini one (at this point the ini file has not yet been read), and define a dummy macro. When the ini file is read, just skip the

corresponding key and reset the macro (in \bbl@inistore above).

```

3556 \def\bbl@renewinikey#1/#2\@#3{%
3557   \edef\bbl@tempa{\zap@space #1 \@empty}%    section
3558   \edef\bbl@tempb{\zap@space #2 \@empty}%    key
3559   \bbl@trim\toks@{#3}%                        value
3560   \bbl@exp{%
3561     \global\let<\bbl@KVP@\bbl@tempa/\bbl@tempb>\@empty % just a flag
3562     \\g@addto@macro\\bbl@inidata{%
3563       \\bbl@elt{\bbl@tempa}{\bbl@tempb}{\the\toks@}}}%

```

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

```

3564 \def\bbl@exportkey#1#2#3{%
3565   \bbl@ifunset{\bbl@kv@#2}%
3566   {\bbl@csarg\gdef{#1@\language}\@empty}%
3567   {\expandafter\ifx\csname\bbl@kv@#2\endcsname\@empty
3568     \bbl@csarg\gdef{#1@\language}\@empty}%
3569   \else
3570     \bbl@exp{\global\let<\bbl@#1@\language><\bbl@kv@#2>}%
3571     \fi}}

```

Key-value pairs are treated differently depending on the section in the ini file. The following macros are the readers for identification and typography. Note \bbl@ini@exports is called always (via \bbl@inisec), while \bbl@after@ini must be called explicitly after \bbl@read@ini if necessary.

```

3572 \def\bbl@iniwarning#1{%
3573   \bbl@ifunset{\bbl@kv@identification.warning#1}{}%
3574   {\bbl@warning{%
3575     From babel-\bbl@cs{lini@\language}.ini:\\
3576     \bbl@cs{@kv@identification.warning#1}\\
3577     Reported }}}
3578 %
3579 \let\bbl@release@transforms\@empty
3580 %
3581 \def\bbl@ini@exports#1{%
3582   % Identification always exported
3583   \bbl@iniwarning}%
3584   \ifcase\bbl@engine
3585     \bbl@iniwarning{.pdflatex}%
3586   \or
3587     \bbl@iniwarning{.lualatex}%
3588   \or
3589     \bbl@iniwarning{.xelatex}%
3590   \fi%
3591   \bbl@exportkey{elname}{identification.name.english}{}%
3592   \bbl@exp{\\bbl@exportkey{lname}{identification.name.opentype}%
3593     {\csname\bbl@elname@\language\endcsname}}%
3594   \bbl@exportkey{tbc}{identification.tag.bcp47}{}%
3595   \bbl@exportkey{lbc}{identification.language.tag.bcp47}{}%
3596   \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
3597   \bbl@exportkey{esname}{identification.script.name}{}%
3598   \bbl@exp{\\bbl@exportkey{sname}{identification.script.name.opentype}%
3599     {\csname\bbl@esname@\language\endcsname}}%
3600   \bbl@exportkey{sbc}{identification.script.tag.bcp47}{}%
3601   \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
3602   % Also maps bcp47 -> language
3603   \ifbbl@bcptoname
3604     \bbl@csarg\xdef{bcp@map@\bbl@cl{tbc}}{\language}%
3605   \fi
3606   % Conditional

```

```

3607 \ifnum#1>\z@          % 0 = only info, 1, 2 = basic, (re)new
3608 \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
3609 \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
3610 \bbl@exportkey{lftm}{typography.lefthyphenmin}{2}%
3611 \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
3612 \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
3613 \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
3614 \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
3615 \bbl@exportkey{intsp}{typography.intraspaces}{}%
3616 \bbl@exportkey{chrng}{characters.ranges}{}%
3617 \bbl@exportkey{dgnat}{numbers.digits.native}{}%
3618 \bbl@exportkey{frspc}{typography.frenchspacing}{u}%
3619 \ifnum#1=\tw@          % only (re)new
3620 \bbl@exportkey{rtex}{identification.require.babel}{}%
3621 \bbl@tglobal\bbl@savetoday
3622 \bbl@tglobal\bbl@savestate
3623 \bbl@savestrings
3624 \fi
3625 \fi}

```

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

```

3626 \def\bbl@inikv#1#2{%      key=value
3627 \toks@{#2}%              This hides #'s from ini values
3628 \bbl@csarg\edef{kv@\bbl@section.#1}{\the\toks@}}

```

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

```

3629 \let\bbl@inikv@identification\bbl@inikv
3630 \let\bbl@inikv@typography\bbl@inikv
3631 \let\bbl@inikv@characters\bbl@inikv
3632 \let\bbl@inikv@numbers\bbl@inikv

```

Additive numerals require an additional definition. When .1 is found, two macros are defined – the basic one, without .1 called by \localnumeral, and another one preserving the trailing .1 for the ‘units’.

```

3633 \def\bbl@inikv@counters#1#2{%
3634 \bbl@ifsamestring{#1}{digits}%
3635 {\bbl@error{The counter name 'digits' is reserved for mapping\\
3636 decimal digits}%
3637 {Use another name.}}%
3638 }%
3639 \def\bbl@tempc{#1}%
3640 \bbl@trim@def{\bbl@tempb*}{#2}%
3641 \in@{.1$}{#1$}%
3642 \ifin@
3643 \bbl@replace\bbl@tempc{.1}{}%
3644 \bbl@csarg\protected@xdef{cntr@\bbl@tempc @\language}%
3645 \noexpand\bbl@alphnumeral{\bbl@tempc}}%
3646 \fi
3647 \in@{.F.}{#1}%
3648 \ifin@ \else \in@{.S.}{#1} \fi
3649 \ifin@
3650 \bbl@csarg\protected@xdef{cntr@#1@\language}{\bbl@tempb*}%
3651 \else
3652 \toks@{ }% Required by \bbl@buildifcase, which returns \bbl@tempa
3653 \expandafter\bbl@buildifcase\bbl@tempb* \ \ % Space after \
3654 \bbl@csarg{\global\expandafter\let}{cntr@#1@\language}\bbl@tempa
3655 \fi}

```

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

that order.

```
3656 \ifcase\bbl@engine
3657   \bbl@csarg\def{inikv@captions.licr}#1#2{%
3658     \bbl@ini@captions@aux{#1}{#2}}
3659 \else
3660   \def\bbl@inikv@captions#1#2{%
3661     \bbl@ini@captions@aux{#1}{#2}}
3662 \fi
```

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

```
3663 \def\bbl@ini@captions@template#1#2{% string language tempa=capt-name
3664   \bbl@replace\bbl@tempa{.template}{}%
3665   \def\bbl@toreplace{#1}{}%
3666   \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace}}%
3667   \bbl@replace\bbl@toreplace{[ ]}{\csname}%
3668   \bbl@replace\bbl@toreplace{[ ]}{\csname the}%
3669   \bbl@replace\bbl@toreplace{[ ]}{name\endcsname}}}%
3670   \bbl@replace\bbl@toreplace{[ ]}{\endcsname}}}%
3671   \bbl@xin@{,\bbl@tempa,}{,chapter,appendix,part,}%
3672   \ifin@
3673     \@nameuse{\bbl@patch\bbl@tempa}%
3674     \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
3675   \fi
3676   \bbl@xin@{,\bbl@tempa,}{,figure,table,}%
3677   \ifin@
3678     \toks@{\expandafter{\bbl@toreplace}}%
3679     \bbl@exp{\gdef\<fnum@\bbl@tempa>{\the\toks@}}%
3680   \fi}
3681 \def\bbl@ini@captions@aux#1#2{%
3682   \bbl@trim@def\bbl@tempa{#1}%
3683   \bbl@xin@{.template}{\bbl@tempa}%
3684   \ifin@
3685     \bbl@ini@captions@template{#2}\languagename
3686   \else
3687     \bbl@ifblank{#2}%
3688     {\bbl@exp{%
3689       \toks@{\bbl@nocaption{\bbl@tempa}{\languagename\bbl@tempa name}}}%
3690     {\bbl@trim\toks@{#2}}}%
3691     \bbl@exp{%
3692       \bbl@add\bbl@savestrings{%
3693         \SetString\<\bbl@tempa name>{\the\toks@}}}%
3694     \toks@{\expandafter{\bbl@captionslist}}%
3695     \bbl@exp{\in@{\<\bbl@tempa name>}{\the\toks@}}%
3696     \ifin@
3697       \bbl@exp{%
3698         \bbl@add\<\bbl@extracaps@\languagename>{\<\bbl@tempa name>}}%
3699         \bbl@tglobal\<\bbl@extracaps@\languagename>}}%
3700     \fi
3701   \fi}
```

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

```
3702 \def\bbl@list@the{%
3703   part,chapter,section,subsection,subsubsection,paragraph,%
3704   subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,%
3705   table,page,footnote,mpfootnote,mpfn}
3706 \def\bbl@map@cnt#1{% #1:roman,etc, // #2:enumi,etc
3707   \bbl@ifunset{\bbl@map@#1\languagename}%
3708   {\@nameuse{#1}}%
3709   {\@nameuse{\bbl@map@#1\languagename}}}
```

```

3710 \def\bbl@inikv@labels#1#2{%
3711   \in@{.map}{#1}%
3712   \ifin@
3713     \ifx\bbl@KVP@labels\@nil\else
3714       \bbl@xin@{ map }{ \bbl@KVP@labels\space}%
3715     \ifin@
3716       \def\bbl@tempc{#1}%
3717       \bbl@replace\bbl@tempc{.map}{}%
3718       \in@{,#2,}{,arabic,roman,Roman,alpha,Alpha,fnsymbol,}%
3719       \bbl@exp{%
3720         \gdef\bbl@map@\bbl@tempc @\language>%
3721         {\ifin@<#2>\else\\localecounter{#2}\fi}}%
3722       \bbl@foreach\bbl@list@the{%
3723         \bbl@ifunset{the##1}{}%
3724         {\bbl@exp{\let\\bbl@tempd\<the##1>}%
3725          \bbl@exp{%
3726            \\bbl@sreplace\<the##1>%
3727            {\<\bbl@tempc>{##1}}{\\\bbl@map@cnt{\bbl@tempc}{##1}}}%
3728            \\bbl@sreplace\<the##1>%
3729            {\<@empty @\bbl@tempc>\<c@##1>}{\\bbl@map@cnt{\bbl@tempc}{##1}}}%
3730          \expandafter\ifx\csname the##1\endcsname\bbl@tempd\else
3731            \toks@ \expandafter\expandafter\expandafter{%
3732              \csname the##1\endcsname}%
3733            \expandafter\xdef\csname the##1\endcsname{{\the\toks@}}%
3734          \fi}}%
3735       \fi
3736     \fi
3737   %
3738   \else
3739     %
3740     % The following code is still under study. You can test it and make
3741     % suggestions. Eg, enumerate.2 = ([enumi]).([enumii]). It's
3742     % language dependent.
3743     \in@{enumerate.}{#1}%
3744     \ifin@
3745       \def\bbl@tempa{#1}%
3746       \bbl@replace\bbl@tempa{enumerate.}{}%
3747       \def\bbl@toreplace{#2}%
3748       \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
3749       \bbl@replace\bbl@toreplace{{}}{\csname the}%
3750       \bbl@replace\bbl@toreplace{{}}{\endcsname{}}%
3751       \toks@ \expandafter{\bbl@toreplace}%
3752       \bbl@exp{%
3753         \\bbl@add\<extras\language>{%
3754           \\babel@save\<labelenum\romannumeral\bbl@tempa>%
3755           \def\<labelenum\romannumeral\bbl@tempa>\the\toks@}%
3756         \\bbl@tglobal\<extras\language>}%
3757       \fi
3758     \fi}

```

To show correctly some captions in a few languages, we need to patch some internal macros, because the order is hardcoded. For example, in Japanese the chapter number is surrounded by two string, while in Hungarian is placed after. These replacement works in many classes, but not all. Actually, the following lines are somewhat tentative.

```

3759 \def\bbl@chapttype{chapter}
3760 \ifx\@makechapterhead\@undefined
3761   \let\bbl@patchchapter\relax
3762 \else\ifx\thechapter\@undefined
3763   \let\bbl@patchchapter\relax

```

```

3764 \elseifx\ps@headings\@undefined
3765 \let\bbl@patchchapter\relax
3766 \else
3767 \def\bbl@patchchapter{%
3768 \global\let\bbl@patchchapter\relax
3769 \bbl@add\appendix{\def\bbl@chapttype{appendix}}% Not harmful, I hope
3770 \bbl@toglobal\appendix
3771 \bbl@sreplace\ps@headings
3772 {\@chapapp\ \thechapter}%
3773 {\bbl@chapterformat}%
3774 \bbl@toglobal\ps@headings
3775 \bbl@sreplace\chaptermark
3776 {\@chapapp\ \thechapter}%
3777 {\bbl@chapterformat}%
3778 \bbl@toglobal\chaptermark
3779 \bbl@sreplace\@makechapterhead
3780 {\@chapapp\space\thechapter}%
3781 {\bbl@chapterformat}%
3782 \bbl@toglobal\@makechapterhead
3783 \gdef\bbl@chapterformat{%
3784 \bbl@ifunset{\bbl@bbl@chapttype fmt@\language}%
3785 {\@chapapp\space\thechapter}
3786 {\@nameuse{\bbl@bbl@chapttype fmt@\language}}}}
3787 \let\bbl@patchappendix\bbl@patchchapter
3788 \fi\fi\fi
3789 \ifx\@part\@undefined
3790 \let\bbl@patchpart\relax
3791 \else
3792 \def\bbl@patchpart{%
3793 \global\let\bbl@patchpart\relax
3794 \bbl@sreplace\@part
3795 {\@partname\nobreakspace\thepart}%
3796 {\bbl@partformat}%
3797 \bbl@toglobal\@part
3798 \gdef\bbl@partformat{%
3799 \bbl@ifunset{\bbl@partfmt@\language}%
3800 {\@partname\nobreakspace\thepart}
3801 {\@nameuse{\bbl@partfmt@\language}}}}
3802 \fi

Date. TODO. Document

3803 % Arguments are _not_ protected.
3804 \let\bbl@calendar\@empty
3805 \DeclareRobustCommand\localedate[1][\bbl@localedate{#1}]
3806 \def\bbl@localedate#1#2#3#4{%
3807 \begingroup
3808 \ifx\@empty#1\@empty\else
3809 \let\bbl@ld@calendar\@empty
3810 \let\bbl@ld@variant\@empty
3811 \edef\bbl@tempa{\zap@space#1 \@empty}%
3812 \def\bbl@tempb##1=##2@@{\@namedef{\bbl@ld@##1}{##2}}%
3813 \bbl@foreach\bbl@tempa{\bbl@tempb##1\@@}%
3814 \edef\bbl@calendar{%
3815 \bbl@ld@calendar
3816 \ifx\bbl@ld@variant\@empty\else
3817 .\bbl@ld@variant
3818 \fi}%
3819 \bbl@replace\bbl@calendar{gregorian}}%
3820 \fi

```

```

3821 \bbl@cased
3822 {\@nameuse\bbl@date@\languagename @\bbl@calendar}{#2}{#3}{#4}}%
3823 \endgroup}
3824 % eg: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
3825 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
3826 \bbl@trim@def\bbl@tempa{#1.#2}%
3827 \bbl@ifsamestring{\bbl@tempa}{months.wide}% to savedate
3828 {\bbl@trim@def\bbl@tempa{#3}%
3829 \bbl@trim\toks@{#5}%
3830 \@temptokena\expandafter{\bbl@savestate}%
3831 \bbl@exp{% Reverse order - in ini last wins
3832 \def\\bbl@savestate{%
3833 \\SetString\<month\romannumeral\bbl@tempa#6name>{\the\toks@}%
3834 \the\@temptokena}}}%
3835 {\bbl@ifsamestring{\bbl@tempa}{date.long}% defined now
3836 {\lowercase{\def\bbl@tempb{#6}}}%
3837 \bbl@trim@def\bbl@toreplace{#5}%
3838 \bbl@TG@@date
3839 \bbl@ifunset\bbl@date@\languagename @}%
3840 {\global\bbl@csarg\let{date@\languagename @}\bbl@toreplace
3841 % TODO. Move to a better place.
3842 \bbl@exp{%
3843 \gdef\<\languagename date>{\protect\<\languagename date >}%
3844 \gdef\<\languagename date >####1####2####3{%
3845 \\bbl@usedategroupttrue
3846 \<bbl@ensure@\languagename>{%
3847 \\localedate{####1}{####2}{####3}}}%
3848 \\bbl@add\\bbl@savetoday{%
3849 \\SetString\\today{%
3850 \<\languagename date>%
3851 {\the\year}{\the\month}{\the\day}}}%
3852 {}%
3853 \ifx\bbl@tempb\@empty\else
3854 \global\bbl@csarg\let{date@\languagename @}\bbl@tempb\bbl@toreplace
3855 \fi}%
3856 {}}}}

```

**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.

```

3857 \let\bbl@calendar\@empty
3858 \newcommand\BabelDateSpace{\nobreakspace}
3859 \newcommand\BabelDateDot{.\@} % TODO. \let instead of repeating
3860 \newcommand\BabelDated[1]{\number#1}
3861 \newcommand\BabelDatedd[1]{\ifnum#1<10 0\fi\number#1}
3862 \newcommand\BabelDateM[1]{\number#1}
3863 \newcommand\BabelDateMM[1]{\ifnum#1<10 0\fi\number#1}
3864 \newcommand\BabelDateMMMM[1]{%
3865 \csname month\romannumeral#1\bbl@calendar name\endcsname}}%
3866 \newcommand\BabelDatey[1]{\number#1}%
3867 \newcommand\BabelDateyy[1]{%
3868 \ifnum#1<10 0\number#1 %
3869 \else\ifnum#1<100 \number#1 %
3870 \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3871 \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %
3872 \else
3873 \bbl@error
3874 {Currently two-digit years are restricted to the\
3875 range 0-9999.}%

```

```

3876      {There is little you can do. Sorry.}%
3877      \fi\fi\fi\fi}
3878 \newcommand\BabelDateyyy[1]{\number#1} % FIXME - add leading 0
3879 \def\bbl@replace@finish@iii#1{%
3880   \bbl@exp{\def\#1###1###2###3{\the\toks@}}
3881 \def\bbl@TG@date{%
3882   \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace{}}%
3883   \bbl@replace\bbl@toreplace{[.]}{\BabelDateDot{}}%
3884   \bbl@replace\bbl@toreplace{[d]}{\BabelDated{###3}}%
3885   \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{###3}}%
3886   \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{###2}}%
3887   \bbl@replace\bbl@toreplace{[MM]}{\BabelDateMM{###2}}%
3888   \bbl@replace\bbl@toreplace{[MMM]}{\BabelDateMMM{###2}}%
3889   \bbl@replace\bbl@toreplace{[y]}{\BabelDatey{###1}}%
3890   \bbl@replace\bbl@toreplace{[yy]}{\BabelDateyy{###1}}%
3891   \bbl@replace\bbl@toreplace{[yyy]}{\BabelDateyyy{###1}}%
3892   \bbl@replace\bbl@toreplace{[y]}{\bbl@datecctr[###1]}%
3893   \bbl@replace\bbl@toreplace{[m]}{\bbl@datecctr[###2]}%
3894   \bbl@replace\bbl@toreplace{[d]}{\bbl@datecctr[###3]}%
3895 % Note after \bbl@replace \toks@ contains the resulting string.
3896 % TODO - Using this implicit behavior doesn't seem a good idea.
3897   \bbl@replace@finish@iii\bbl@toreplace}
3898 \def\bbl@datecctr{\expandafter\bbl@xdatecctr\expandafter}
3899 \def\bbl@xdatecctr[#1|#2]{\localnumeral{#2}{#1}}

```

#### Transforms.

```

3900 \let\bbl@release@transforms\empty
3901 \@namedef{bbl@inikv@transforms.prehyphenation}{%
3902   \bbl@transforms\babelprehyphenation}
3903 \@namedef{bbl@inikv@transforms.posthyphenation}{%
3904   \bbl@transforms\babelposthyphenation}
3905 \def\bbl@transforms@aux#1#2#3,#4\relax{#1{#2}{#3}{#4}}
3906 \begingroup
3907   \catcode`\%=12
3908   \catcode`\&=14
3909   \gdef\bbl@transforms#1#2#3{&%
3910     \ifx\bbl@KVP@transforms\@nil\else
3911       \directlua{
3912         str = [=[#2]=]
3913         str = str:gsub('%.%d+%.%d+$', '')
3914         tex.print([[def\string\babeltempa{]] .. str .. [[]])
3915       }&%
3916       \bbl@xin@{,\babeltempa,},{,\bbl@KVP@transforms,}&%
3917       \ifin@
3918         \in@{.0$}{#2$}&%
3919       \ifin@
3920         \g@addto@macro\bbl@release@transforms{&%
3921           \relax\bbl@transforms@aux#1{\language}\{#3}}&%
3922       \else
3923         \g@addto@macro\bbl@release@transforms{, {#3}}&%
3924       \fi
3925     \fi
3926   \fi}
3927 \endgroup

```

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

```

3928 \def\bbl@provide@lsys#1{%
3929   \bbl@ifunset{bbl@lname@#1}%

```



```

3930 {\bbl@load@info{#1}}%
3931 {}%
3932 \bbl@csarg\let{lsys@#1}\@empty
3933 \bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}{}%
3934 \bbl@ifunset{bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{}%
3935 \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
3936 \bbl@ifunset{bbl@lname@#1}{}%
3937 {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
3938 \ifcase\bbl@engine\or\or
3939 \bbl@ifunset{bbl@prehc@#1}{}%
3940 {\bbl@exp{\bbl@ifblank{\bbl@cs{prehc@#1}}}%
3941 {}%
3942 {\ifx\bbl@xenoxyph\@undefined
3943 \let\bbl@xenoxyph\bbl@xenoxyph@d
3944 \ifx\AtBeginDocument\@notprerr
3945 \expandafter\@secondoftwo % to execute right now
3946 \fi
3947 \AtBeginDocument{%
3948 \expandafter\bbl@add
3949 \csname selectfont \endcsname{\bbl@xenoxyph}%
3950 \expandafter\selectlanguage\expandafter{\language}%
3951 \expandafter\bbl@together\csname selectfont \endcsname}%
3952 \fi}}%
3953 \fi
3954 \bbl@csarg\bbl@together{lsys@#1}}
3955 \def\bbl@xenoxyph@d{%
3956 \bbl@ifset{bbl@prehc@language}%
3957 {\ifnum\hyphenchar\font=\defaultshyphenchar
3958 \iffontchar\font\bbl@c1{prehc}\relax
3959 \hyphenchar\font\bbl@c1{prehc}\relax
3960 \else\iffontchar\font"200B
3961 \hyphenchar\font"200B
3962 \else
3963 \bbl@warning
3964 {Neither 0 nor ZERO WIDTH SPACE are available\\%
3965 in the current font, and therefore the hyphen\\%
3966 will be printed. Try changing the fontspec's\\%
3967 'HyphenChar' to another value, but be aware\\%
3968 this setting is not safe (see the manual)}%
3969 \hyphenchar\font\defaultshyphenchar
3970 \fi\fi
3971 \fi}%
3972 {\hyphenchar\font\defaultshyphenchar}}
3973 % \fi}

```

The following ini reader ignores everything but the identification section. It is called when a font is defined (ie, when the language is first selected) to know which script/language must be enabled. This means we must make sure a few characters are not active. The ini is not read directly, but with a proxy tex file named as the language (which means any code in it must be skipped, too).

```

3974 \def\bbl@load@info#1{%
3975 \def\BabelBeforeIni##1##2{%
3976 \begingroup
3977 \bbl@read@ini{##1}0%
3978 \endinput % babel- .tex may contain only preamble's
3979 \endgroup}% boxed, to avoid extra spaces:
3980 {\bbl@input@texini{#1}}}

```

A tool to define the macros for native digits from the list provided in the ini file. Somewhat convoluted because there are 10 digits, but only 9 arguments in T<sub>E</sub>X. Non-digits characters are kept.

[illegible]

```

4012 \def\bbl@buildifcase#1 {% Returns \bbl@tempa, requires \toks@={}%
4013   \ifx\\#1%           % \\ before, in case #1 is multiletter
4014     \bbl@exp{%
4015       \def\\ \bbl@tempa####1{%
4016         \<ifcase>####1\space\the\toks@\<else>\\ \ctrerr\<fi>}}%
4017   \else
4018     \toks@\expandafter{\the\toks@\or #1}%
4019     \expandafter\bbl@buildifcase
4020 \fi}

```

```

4021 \newcommand\localenumeral[2]{\bbl@cs{cntr@#1\language}\csname c@#2\endcsname}{#2}}
4022 \def\bbl@localecntr#1#2{\localenumeral{#2}{#1}}
4023 \newcommand\localecounter[2]{%
4024   \expandafter\bbl@localecntr
4025   \expandafter{\number\csname c@#2\endcsname}{#1}}
4026 \def\bbl@alphnumeral#1#2{%
4027   \expandafter\bbl@alphnumeral@i\number#2 76543210\@@{#1}}
4028 \def\bbl@alphnumeral@i#1#2#3#4#5#6#7#8\@@#9{%
4029   \ifcase\car#8\nilor % Currently <10000, but prepared for bigger
4030     \bbl@alphnumeral@ii{#9}000000#1\or

```

```

4031 \bbl@alphanumeric@ii{#9}0000#1#2\or
4032 \bbl@alphanumeric@ii{#9}0000#1#2#3\or
4033 \bbl@alphanumeric@ii{#9}000#1#2#3#4\else
4034 \bbl@alphnum@invalid{>9999}%
4035 \fi}
4036 \def\bbl@alphanumeric@ii#1#2#3#4#5#6#7#8{%
4037 \bbl@ifunset{bbl@cntr@#1.F.\number#5#6#7#8@\language}%
4038 {\bbl@cs{cntr@#1.4@\language}#5%
4039 \bbl@cs{cntr@#1.3@\language}#6%
4040 \bbl@cs{cntr@#1.2@\language}#7%
4041 \bbl@cs{cntr@#1.1@\language}#8%
4042 \ifnum#6#7#8>\z@ % TODO. An ad hoc rule for Greek. Ugly.
4043 \bbl@ifunset{bbl@cntr@#1.S.321@\language}{}}%
4044 {\bbl@cs{cntr@#1.S.321@\language}}%
4045 \fi}%
4046 {\bbl@cs{cntr@#1.F.\number#5#6#7#8@\language}}%
4047 \def\bbl@alphnum@invalid#1{%
4048 \bbl@error{Alphabetic numeral too large (#1)}%
4049 {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.

```

4050 \newcommand\localeinfo[1]{%
4051 \bbl@ifunset{bbl@csname bbl@info@#1\endcsname @\language}%
4052 {\bbl@error{I've found no info for the current locale.\%
4053 The corresponding ini file has not been loaded\%
4054 Perhaps it doesn't exist}%
4055 {See the manual for details.}}%
4056 {\bbl@cs{csname bbl@info@#1\endcsname @\language}}%
4057 \@namedef{bbl@info@name.locale}{lcname}
4058 \@namedef{bbl@info@tag.ini}{lini}
4059 \@namedef{bbl@info@name.english}{elname}
4060 \@namedef{bbl@info@name.opentype}{lname}
4061 \@namedef{bbl@info@tag.bcp47}{tbc}
4062 \@namedef{bbl@info@language.tag.bcp47}{lbcp}
4063 \@namedef{bbl@info@tag.opentype}{lotf}
4064 \@namedef{bbl@info@script.name}{esname}
4065 \@namedef{bbl@info@script.name.opentype}{sname}
4066 \@namedef{bbl@info@script.tag.bcp47}{sbcp}
4067 \@namedef{bbl@info@script.tag.opentype}{sotf}
4068 \let\bbl@ensureinfo\@gobble
4069 \newcommand\BabelEnsureInfo{%
4070 \ifx\InputIfFileExists\undefined\else
4071 \def\bbl@ensureinfo##1{%
4072 \bbl@ifunset{bbl@lname@##1}{\bbl@load@info{##1}}}%
4073 \fi
4074 \bbl@foreach\bbl@loaded{%
4075 \def\language{##1}%
4076 \bbl@ensureinfo{##1}}}

```

More general, but non-expandable, is `\getlocaleproperty`. To inspect every possible loaded ini, we define `\LocaleForEach`, where `\bbl@ini@loaded` is a comma-separated list of locales, built by `\bbl@read@ini`.

```

4077 \newcommand\getlocaleproperty{%
4078 \@ifstar\bbl@getproperty@s\bbl@getproperty@x}
4079 \def\bbl@getproperty@s#1#2#3{%
4080 \let#1\relax
4081 \def\bbl@elt##1##2##3{%
4082 \bbl@ifsamestring{##1/##2}{##3}%

```

```

4083     {\providecommand#1{##3}%
4084     \def\bbl@elt####1####2####3{}}}%
4085     {}}}%
4086     \bbl@cs{inidata@#2}}}%
4087 \def\bbl@getproperty@x#1#2#3{%
4088     \bbl@getproperty@s{#1}{#2}{#3}%
4089     \ifx#1\relax
4090         \bbl@error
4091         {Unknown key for locale '#2':\%
4092         #3\}%
4093         \string#1 will be set to \relax}%
4094     {Perhaps you misspelled it.}%
4095     \fi}
4096 \let\bbl@ini@loaded\@empty
4097 \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.

```

4098 \newcommand\babeladjust[1]{% TODO. Error handling.
4099     \bbl@forkv{#1}{%
4100         \bbl@ifunset{\bbl@ADJ@##1@##2}%
4101         {\bbl@cs{ADJ@##1}{##2}}}%
4102         {\bbl@cs{ADJ@##1@##2}}}}
4103 %
4104 \def\bbl@adjust@lua#1#2{%
4105     \ifvmode
4106         \ifnum\currentgrouplevel=\z@
4107             \directlua{ Babel.#2 }%
4108             \expandafter\expandafter\expandafter\@gobble
4109             \fi
4110         \fi
4111         {\bbl@error % The error is gobbled if everything went ok.
4112         {Currently, #1 related features can be adjusted only\%
4113         in the main vertical list.}%
4114         {Maybe things change in the future, but this is what it is.}}}
4115 \@namedef{\bbl@ADJ@bidi.mirroring@on}{%
4116     \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
4117 \@namedef{\bbl@ADJ@bidi.mirroring@off}{%
4118     \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
4119 \@namedef{\bbl@ADJ@bidi.text@on}{%
4120     \bbl@adjust@lua{bidi}{bidi_enabled=true}}
4121 \@namedef{\bbl@ADJ@bidi.text@off}{%
4122     \bbl@adjust@lua{bidi}{bidi_enabled=false}}
4123 \@namedef{\bbl@ADJ@bidi.mapdigits@on}{%
4124     \bbl@adjust@lua{bidi}{digits_mapped=true}}
4125 \@namedef{\bbl@ADJ@bidi.mapdigits@off}{%
4126     \bbl@adjust@lua{bidi}{digits_mapped=false}}
4127 %
4128 \@namedef{\bbl@ADJ@linebreak.sea@on}{%
4129     \bbl@adjust@lua{linebreak}{sea_enabled=true}}
4130 \@namedef{\bbl@ADJ@linebreak.sea@off}{%
4131     \bbl@adjust@lua{linebreak}{sea_enabled=false}}
4132 \@namedef{\bbl@ADJ@linebreak.cjk@on}{%
4133     \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
4134 \@namedef{\bbl@ADJ@linebreak.cjk@off}{%
4135     \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
4136 %

```

```

4137 \def\bbl@adjust@layout#1{%
4138   \ifvmode
4139     #1%
4140   \expandafter\@gobble
4141   \fi
4142   {\bbl@error   % The error is gobbled if everything went ok.
4143     {Currently, layout related features can be adjusted only\%
4144       in vertical mode.}%
4145     {Maybe things change in the future, but this is what it is.}}}
4146 \@namedef{bbl@ADJ@layout.tabular@on}{%
4147   \bbl@adjust@layout{\let\@tabular\bbl@NL@tabular}}
4148 \@namedef{bbl@ADJ@layout.tabular@off}{%
4149   \bbl@adjust@layout{\let\@tabular\bbl@OL@tabular}}
4150 \@namedef{bbl@ADJ@layout.lists@on}{%
4151   \bbl@adjust@layout{\let\list\bbl@NL@list}}
4152 \@namedef{bbl@ADJ@layout.lists@off}{%
4153   \bbl@adjust@layout{\let\list\bbl@OL@list}}
4154 \@namedef{bbl@ADJ@hyphenation.extra@on}{%
4155   \bbl@activateposthyphen}
4156 %
4157 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
4158   \bbl@bcpallowedtrue}
4159 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
4160   \bbl@bcpallowedfalse}
4161 \@namedef{bbl@ADJ@autoload.bcp47.prefix#1}{%
4162   \def\bbl@bcp@prefix{#1}}
4163 \def\bbl@bcp@prefix{bcp47-}
4164 \@namedef{bbl@ADJ@autoload.options#1}{%
4165   \def\bbl@autoload@bcptoptions{#1}}
4166 \let\bbl@autoload@bcptoptions\@empty
4167 \@namedef{bbl@ADJ@autoload.bcp47.options#1}{%
4168   \def\bbl@autoload@bcptoptions{#1}}
4169 \newif\ifbbl@bcptoname
4170 \@namedef{bbl@ADJ@bcp47.toname@on}{%
4171   \bbl@bcptonametrue}
4172 \BabelEnsureInfo}
4173 \@namedef{bbl@ADJ@bcp47.toname@off}{%
4174   \bbl@bcptonamefalse}
4175 % TODO: use babel name, override
4176 %
4177 % As the final task, load the code for lua.
4178 %
4179 \ifx\directlua\@undefined\else
4180   \ifx\bbl@luapatterns\@undefined
4181     \input luababel.def
4182   \fi
4183 \fi
4184 </core>

A proxy file for switch.def

4185 <*kernel>
4186 \let\bbl@onlyswitch\@empty
4187 \input babel.def
4188 \let\bbl@onlyswitch\@undefined
4189 </kernel>
4190 <*patterns>

```

## 11 Loading hyphenation patterns

The following code is meant to be read by  $\text{\texttt{iniT\TeX}}$  because it should instruct  $\text{\texttt{T\TeX}}$  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  $\text{\texttt{L\TeX}}$  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.

```

4191 <<Make sure ProvidesFile is defined>>
4192 \ProvidesFile{hyphen.cfg}[\<date>] [\<version>] Babel hyphens]
4193 \xdef\bbl@format{\jobname}
4194 \def\bbl@version{\<version>}
4195 \def\bbl@date{\<date>}
4196 \ifx\AtBeginDocument\@undefined
4197   \def\@empty{}
4198   \let\orig@dump\dump
4199   \def\dump{%
4200     \ifx\@ztryfc\@undefined
4201       \else
4202         \toks0=\expandafter{\@preamblecmds}%
4203         \edef\@preamblecmds{\noexpand\@begindocumenthook\the\toks0}%
4204         \def\@begindocumenthook{}%
4205       \fi
4206       \let\dump\orig@dump\let\orig@dump\@undefined\dump}
4207 \fi
4208 <<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.

```

4209 \def\process@line#1#2 #3 #4 {%
4210   \ifx=#1%
4211     \process@synonym{#2}%
4212   \else
4213     \process@language{#1#2}{#3}{#4}%
4214   \fi
4215   \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.

```

4216 \toks@{}
4217 \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.

```

4218 \def\process@synonym#1{%
4219   \ifnum\last@language=\m@ne
4220     \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4221   \else
4222     \expandafter\chardef\csname l@#1\endcsname\last@language
4223     \wlog{\string\l@#1=\string\language\the\last@language}%

```

```

4224 \expandafter\let\csname #1hyphenmins\expandafter\endcsname
4225 \csname\language\language\hyphenmins\endcsname
4226 \let\bb1@elt\relax
4227 \edef\bb1@languages{\bb1@languages\bb1@elt{#1}{\the\last@language}{}}}%
4228 \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 `\bb1@get@enc` extracts the font encoding from the language name and stores it in `\bb1@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`. T<sub>E</sub>X 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.)

`\bb1@languages` saves a snapshot of the loaded languages in the form

`\bb1@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.

```

4229 \def\process@language#1#2#3{%
4230 \expandafter\addlanguage\csname l@#1\endcsname
4231 \expandafter\language\csname l@#1\endcsname
4232 \edef\language{#1}%
4233 \bb1@hook@everylanguage{#1}%
4234 % > luatex
4235 \bb1@get@enc#1::\@@@
4236 \begingroup
4237 \lefthyphenmin\m@ne
4238 \bb1@hook@loadpatterns{#2}%
4239 % > luatex
4240 \ifnum\lefthyphenmin=\m@ne
4241 \else
4242 \expandafter\xdef\csname #1hyphenmins\endcsname{%
4243 \the\lefthyphenmin\the\righthyphenmin}%
4244 \fi
4245 \endgroup
4246 \def\bb1@tempa{#3}%
4247 \ifx\bb1@tempa\@empty\else
4248 \bb1@hook@loadexceptions{#3}%
4249 % > luatex
4250 \fi
4251 \let\bb1@elt\relax
4252 \edef\bb1@languages{%
4253 \bb1@languages\bb1@elt{#1}{\the\language}{#2}{\bb1@tempa}}%
4254 \ifnum\the\language=\z@

```

```

4255 \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4256 \set@hyphenmins\tw@thr@@\relax
4257 \else
4258 \expandafter\expandafter\expandafter\set@hyphenmins
4259 \csname #1hyphenmins\endcsname
4260 \fi
4261 \the\toks@
4262 \toks@{}%
4263 \fi}

```

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

```

4264 \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.

```

4265 \def\bbl@hook@everylanguage#1{}
4266 \def\bbl@hook@loadpatterns#1{\input #1\relax}
4267 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
4268 \def\bbl@hook@loadkernel#1{%
4269 \def\addlanguage{\csname newlanguage\endcsname}%
4270 \def\adddialect##1##2{%
4271 \global\chardef##1##2\relax
4272 \wlog{\string##1 = a dialect from \string\language##2}}%
4273 \def\iflanguage##1{%
4274 \expandafter\ifx\csname l@##1\endcsname\relax
4275 \nolannerr{##1}%
4276 \else
4277 \ifnum\csname l@##1\endcsname=\language
4278 \expandafter\expandafter\expandafter\@firstoftwo
4279 \else
4280 \expandafter\expandafter\expandafter\@secondoftwo
4281 \fi
4282 \fi}%
4283 \def\providehyphenmins##1##2{%
4284 \expandafter\ifx\csname ##1hyphenmins\endcsname\relax
4285 \@namedef{##1hyphenmins}{##2}%
4286 \fi}%
4287 \def\set@hyphenmins##1##2{%
4288 \lefthyphenmin##1\relax
4289 \righthyphenmin##2\relax}%
4290 \def\selectlanguage{%
4291 \errhelp{Selecting a language requires a package supporting it}%
4292 \errmessage{Not loaded}}%
4293 \let\foreignlanguage\selectlanguage
4294 \let\otherlanguage\selectlanguage
4295 \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4296 \def\bbl@usehooks##1##2{% TODO. Temporary!!
4297 \def\setlocale{%
4298 \errhelp{Find an armchair, sit down and wait}%
4299 \errmessage{Not yet available}}%
4300 \let\uselocale\setlocale
4301 \let\locale\setlocale
4302 \let\selectlocale\setlocale
4303 \let\localename\setlocale
4304 \let\textlocale\setlocale
4305 \let\textlanguage\setlocale

```



```

4306 \let\languagetext\setlocale}
4307 \begingroup
4308 \def\AddBabelHook#1#2{%
4309   \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
4310     \def\next{\toks1}%
4311   \else
4312     \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname####1}%
4313   \fi
4314   \next}
4315 \ifx\directlua@undefined
4316   \ifx\XeTeXinputencoding@undefined\else
4317     \input xebabel.def
4318   \fi
4319 \else
4320   \input luababel.def
4321 \fi
4322 \openin1 = babel-\bbl@format.cfg
4323 \ifeof1
4324 \else
4325   \input babel-\bbl@format.cfg\relax
4326 \fi
4327 \closein1
4328 \endgroup
4329 \bbl@hook@loadkernel{switch.def}

```

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

```

4330 \openin1 = language.dat

```

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

```

4331 \def\language{english}%
4332 \ifeof1
4333   \message{I couldn't find the file language.dat,\space
4334             I will try the file hyphen.tex}
4335   \input hyphen.tex\relax
4336   \chardef\l@english\z@
4337 \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  $-1$ .

```

4338 \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.

```

4339 \loop
4340   \endlinechar\m@ne
4341   \read1 to \bbl@line
4342   \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.

```

4343   \if T\ifeof1F\fi T\relax
4344   \ifx\bbl@line\@empty\else
4345     \edef\bbl@line{\bbl@line\space\space\space}%
4346     \expandafter\process@line\bbl@line\relax

```

```

4347 \fi
4348 \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.

```

4349 \begingroup
4350 \def\bbl@elt#1#2#3#4{%
4351 \global\language=#2\relax
4352 \gdef\language#1}%
4353 \def\bbl@elt##1##2##3##4{}}%
4354 \bbl@languages
4355 \endgroup
4356 \fi
4357 \closein1

```

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

```

4358 \if\the\toks@\else
4359 \errhelp{language.dat loads no language, only synonyms}
4360 \errmessage{Orphan language synonym}
4361 \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.

```

4362 \let\bbl@line\@undefined
4363 \let\process@line\@undefined
4364 \let\process@synonym\@undefined
4365 \let\process@language\@undefined
4366 \let\bbl@get@enc\@undefined
4367 \let\bbl@hyph@enc\@undefined
4368 \let\bbl@tempa\@undefined
4369 \let\bbl@hook@loadkernel\@undefined
4370 \let\bbl@hook@everylanguage\@undefined
4371 \let\bbl@hook@loadpatterns\@undefined
4372 \let\bbl@hook@loadexceptions\@undefined
4373 \patterns)

```

Here the code for `iniTeX` ends.

## 12 Font handling with fontspec

Add the bidi handler just before `luaotfload`, 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].

```

4374 <<(*More package options)>> ≡
4375 \chardef\bbl@bidimode\z@
4376 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4377 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
4378 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4379 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4380 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4381 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4382 <</More package options>>

```

With explicit languages, we could define the font at once, but we don't. Just wait and see if the language is actually activated. `bbl@font` replaces hardcoded font names inside `\. . family` by the corresponding macro `\. . default`.

At the time of this writing, `fontspec` shows a warning about there are languages not available, which some people think refers to `babel`, even if there is nothing wrong. Here is hack to patch `fontspec` to avoid the misleading message, which is replaced by a more explanatory one.

```

4383 <<*Font selection>> ≡
4384 \bbl@trace{Font handling with fontspec}
4385 \ifx\ExplSyntaxOn\@undefined\else
4386   \ExplSyntaxOn
4387   \catcode\ =10
4388   \def\bbl@loadfontspec{%
4389     \usepackage{fontspec}%
4390     \expandafter
4391     \def\csname msg~text~>~fontspec/language-not-exist\endcsname##1##2##3##4{%
4392       Font '\l_fontspec_fontname_tl' is using the\\%
4393       default features for language '##1'.\\%
4394       That's usually fine, because many languages\\%
4395       require no specific features, but if the output is\\%
4396       not as expected, consider selecting another font.}
4397     \expandafter
4398     \def\csname msg~text~>~fontspec/no-script\endcsname##1##2##3##4{%
4399       Font '\l_fontspec_fontname_tl' is using the\\%
4400       default features for script '##2'.\\%
4401       That's not always wrong, but if the output is\\%
4402       not as expected, consider selecting another font.}}
4403   \ExplSyntaxOff
4404 \fi
4405 \@onlypreamble\babelfont
4406 \newcommand\babelfont[2][{% 1=langs/scripts 2=fam
4407   \bbl@foreach{#1}{%
4408     \expandafter\ifx\csname date##1\endcsname\relax
4409       \IfFileExists{babel-##1.tex}%
4410       {\babelprovide{##1}}%
4411       {}%
4412     \fi}%
4413   \edef\bbl@tempa{#1}%
4414   \def\bbl@tempb{#2}% Used by \bbl@bblfont
4415   \ifx\fontspec\@undefined
4416     \bbl@loadfontspec
4417   \fi
4418   \EnableBabelHook{babel-fontspec}% Just calls \bbl@switchfont
4419   \bbl@bblfont}
4420 \newcommand\bbl@bblfont[2][{% 1=features 2=fontname, @font=rm|sf|tt
4421   \bbl@ifunset{\bbl@tempb family}%
4422   {\bbl@providedefam{\bbl@tempb}}%
4423   {\bbl@exp{%
4424     \\\bbl@sreplace\<\bbl@tempb family >%
4425     {\@nameuse{\bbl@tempb default}}{\<\bbl@tempb default>}}}%
4426   % For the default font, just in case:
4427   \bbl@ifunset{\bbl@lsys@languagenam}{\bbl@provide@lsys{languagenam}}}%
4428   \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
4429   {\bbl@csarg\edef{\bbl@tempb dflt@}{<{#1}{#2}}% save bbl@rmdflt@
4430   \bbl@exp{%
4431     \let\<\bbl@tempb dflt@\languagenam>\<\bbl@tempb dflt@>%
4432     \\\bbl@font@set\<\bbl@tempb dflt@\languagenam>%
4433     \<\bbl@tempb default>\<\bbl@tempb family>}}%
4434   {\bbl@foreach\bbl@tempa{ ie bbl@rmdflt@lang / *scrt
4435     \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:

```

4436 \def\bbl@providedefam#1{%
4437   \bbl@exp{%
4438     \\\newcommand\<#1default>{}% Just define it
4439     \\\bbl@add@list\ \bbl@font@fams{#1}%

```

```

4440   \\\DeclareRobustCommand\<#1family>{%
4441     \\\not@math@alphabet\<#1family>\relax
4442     \\\fontfamily\<#1default>\\selectfont}%
4443   \\\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.

```

4444 \def\bbl@nostdfont#1{%
4445   \bbl@ifunset{\bbl@WFF@f@family}%
4446     {\bbl@csarg\gdef{\bbl@WFF@f@family}}{% Flag, to avoid dupl warns
4447     \bbl@infowarn{The current font is not a babel standard family:\%
4448       #1%
4449       \fontname\font\\%
4450       There is nothing intrinsically wrong with this warning, and\\%
4451       you can ignore it altogether if you do not need these\\%
4452       families. But if they are used in the document, you should be\\%
4453       aware 'babel' will no set Script and Language for them, so\\%
4454       you may consider defining a new family with \string\babelfont.\\%
4455       See the manual for further details about \string\babelfont.\\%
4456       Reported}}
4457   }%
4458 \gdef\bbl@switchfont{%
4459   \bbl@ifunset{\bbl@lsys@language}{\bbl@provide@lsys{language}}}%
4460   \bbl@exp{% eg Arabic -> arabic
4461     \lowercase{\edef\bbl@tempa{\bbl@cl{sname}}}%
4462     \bbl@foreach\bbl@font@fams{%
4463       \bbl@ifunset{\bbl@##1dflt@language}% (1) language?
4464       {\bbl@ifunset{\bbl@##1dflt@*\bbl@tempa}% (2) from script?
4465       {\bbl@ifunset{\bbl@##1dflt@}% 2=F - (3) from generic?
4466       }% 123=F - nothing!
4467       {\bbl@exp{% 3=T - from generic
4468         \global\let\<bbl@##1dflt@language>%
4469         \<bbl@##1dflt@>}}}%
4470       {\bbl@exp{% 2=T - from script
4471         \global\let\<bbl@##1dflt@language>%
4472         \<bbl@##1dflt@*\bbl@tempa>}}}%
4473       }% 1=T - language, already defined
4474     \def\bbl@tempa{\bbl@nostdfont}}%
4475     \bbl@foreach\bbl@font@fams{% don't gather with prev for
4476       \bbl@ifunset{\bbl@##1dflt@language}%
4477       {\bbl@cs{famrst@##1}%
4478       \global\bbl@csarg\let{famrst@##1}\relax}%
4479       {\bbl@exp{% order is relevant. TODO: but sometimes wrong!
4480         \\\bbl@add\\originalTeX{%
4481         \\\bbl@font@rst{\bbl@cl{##1dflt}}%
4482         \<##1default>\<##1family>{##1}}%
4483         \\\bbl@font@set\<bbl@##1dflt@language>% the main part!
4484         \<##1default>\<##1family>}}}%
4485     \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.

```

4486 \ifx\fontfamily\undefined\else % if latex
4487 \ifcase\bbl@engine % if pdftex
4488   \let\bbl@cckstdfonts\relax
4489 \else
4490   \def\bbl@cckstdfonts{%
4491     \begingroup
4492     \global\let\bbl@cckstdfonts\relax

```

```

4493 \let\bbl@tempa\@empty
4494 \bbl@foreach\bbl@font@fams{%
4495 \bbl@ifunset{bbl@##1dflt@}%
4496 {\nameuse{##1family}%
4497 \bbl@csarg\gdef{WFF@f@family}{}}% Flag
4498 \bbl@exp{\bbl@add\bbl@tempa{* \<##1family>= f@family\\%
4499 \space\space\fontname\font\\}%
4500 \bbl@csarg\xdef{##1dflt@}{f@family}%
4501 \expandafter\xdef\csname ##1default\endcsname{f@family}}%
4502 {}}%
4503 \ifx\bbl@tempa\@empty\else
4504 \bbl@info{The following font families will use the default\\%
4505 settings for all or some languages:\\%
4506 \bbl@tempa
4507 There is nothing intrinsically wrong with it, but\\%
4508 'babel' will no set Script and Language, which could\\%
4509 be relevant in some languages. If your document uses\\%
4510 these families, consider redefining them with \string\babelfont.\\%
4511 Reported}%
4512 \fi
4513 \endgroup}
4514 \fi
4515 \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.

```

4516 \def\bbl@font@set#1#2#3{% eg \bbl@rmdflt@lang \rmdefault \rmfamily
4517 \bbl@xin@{<>}{#1}%
4518 \ifin@
4519 \bbl@exp{\bbl@fontspec@set\#1\expandafter@gobbletwo#1\#3}%
4520 \fi
4521 \bbl@exp{% 'Unprotected' macros return prev values
4522 \def\#2{#1}% eg, \rmdefault{\bbl@rmdflt@lang}
4523 \bbl@ifsamestring{#2}{f@family}%
4524 {\#3%
4525 \bbl@ifsamestring{f@series}{bfdefault}{\bfseries}}%
4526 \let\bbl@tempa\relax}%
4527 {}}}
4528 % TODO - next should be global?, but even local does its job. I'm
4529 % still not sure -- must investigate:
4530 \def\bbl@fontspec@set#1#2#3#4{% eg \bbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
4531 \let\bbl@tempa\bbl@mapselect
4532 \let\bbl@mapselect\relax
4533 \let\bbl@temp@fam#4% eg, '\rmfamily', to be restored below
4534 \let#4\@empty % Make sure \renewfontfamily is valid
4535 \bbl@exp{%
4536 \let\bbl@temp@pfam\<\bbl@stripslash#4\space>% eg, '\rmfamily '
4537 \<keys_if_exist:nnF>{fontspec-opentype}{Script/\bbl@cl{sname}}%
4538 {\newfontscript{\bbl@cl{sname}}{\bbl@cl{sotf}}}%
4539 \<keys_if_exist:nnF>{fontspec-opentype}{Language/\bbl@cl{lname}}%
4540 {\newfontlanguage{\bbl@cl{lname}}{\bbl@cl{lotf}}}%
4541 \renewfontfamily\#4%
4542 [\bbl@cs{lsys@\languagename},#2]}{#3}% ie \bbl@exp{.}{#3}
4543 \begingroup
4544 #4%
4545 \xdef#1{f@family}% eg, \bbl@rmdflt@lang{FreeSerif(0)}
4546 \endgroup

```

```

4547 \let#4\bb1@temp@fam
4548 \bb1@exp{\let\<\bb1@stripslash#4\space>}\bb1@temp@pfam
4549 \let\bb1@mapselect\bb1@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.

```

4550 \def\bb1@font@rst#1#2#3#4{%
4551 \bb1@csarg\def{famrst@#4}{\bb1@font@set{#1}#2#3}}

```

The default font families. They are eurocentric, but the list can be expanded easily with \babelfont.

```

4552 \def\bb1@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 :-).

```

4553 \newcommand\babelFSstore[2][]{%
4554 \bb1@ifblank{#1}%
4555 {\bb1@csarg\def{sname@#2}{Latin}}}%
4556 {\bb1@csarg\def{sname@#2}{#1}}}%
4557 \bb1@provide@dirs{#2}%
4558 \bb1@csarg\ifnum{wdir@#2}>\z@
4559 \let\bb1@beforeforeign\leavevmode
4560 \EnableBabelHook{babel-bidi}%
4561 \fi
4562 \bb1@foreach{#2}{%
4563 \bb1@FSstore{##1}{rm}\rmdefault\bb1@save@rmdefault
4564 \bb1@FSstore{##1}{sf}\sfdefault\bb1@save@sfdefault
4565 \bb1@FSstore{##1}{tt}\ttdefault\bb1@save@ttdefault}}
4566 \def\bb1@FSstore#1#2#3#4{%
4567 \bb1@csarg\edef{#2default#1}{#3}%
4568 \expandafter\addto\csname extras#1\endcsname{%
4569 \let#4#3%
4570 \ifx#3\fbfamily
4571 \edef#3{\csname bbl@#2default#1\endcsname}%
4572 \fontfamily{#3}\selectfont
4573 \else
4574 \edef#3{\csname bbl@#2default#1\endcsname}%
4575 \fi}%
4576 \expandafter\addto\csname noextras#1\endcsname{%
4577 \ifx#3\fbfamily
4578 \fontfamily{#4}\selectfont
4579 \fi
4580 \let#3#4}}
4581 \let\bb1@langfeatures\@empty
4582 \def\babelFSfeatures{% make sure \fontspec is redefined once
4583 \let\bb1@ori@fontspec\fontspec
4584 \renewcommand\fontspec[1][]{%
4585 \bb1@ori@fontspec[\bb1@langfeatures##1]}
4586 \let\babelFSfeatures\bb1@FSfeatures
4587 \babelFSfeatures}
4588 \def\bb1@FSfeatures#1#2{%
4589 \expandafter\addto\csname extras#1\endcsname{%
4590 \babel@save\bb1@langfeatures
4591 \edef\bb1@langfeatures{#2,}}
4592 <</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.

```
4593 <<*Footnote changes>> ≡
4594 \bbl@trace{Bidi footnotes}
4595 \ifnum\bbl@bidimode>\z@
4596   \def\bbl@footnote#1#2#3{%
4597     \ifnextchar[%
4598       {\bbl@footnote@o{#1}{#2}{#3}}%
4599       {\bbl@footnote@x{#1}{#2}{#3}}}
4600   \long\def\bbl@footnote@x#1#2#3#4{%
4601     \bgroup
4602     \select@language@x{\bbl@main@language}%
4603     \bbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
4604     \egroup}
4605   \long\def\bbl@footnote@o#1#2#3[#4]#5{%
4606     \bgroup
4607     \select@language@x{\bbl@main@language}%
4608     \bbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
4609     \egroup}
4610   \def\bbl@footnotetext#1#2#3{%
4611     \ifnextchar[%
4612       {\bbl@footnotetext@o{#1}{#2}{#3}}%
4613       {\bbl@footnotetext@x{#1}{#2}{#3}}}
4614   \long\def\bbl@footnotetext@x#1#2#3#4{%
4615     \bgroup
4616     \select@language@x{\bbl@main@language}%
4617     \bbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
4618     \egroup}
4619   \long\def\bbl@footnotetext@o#1#2#3[#4]#5{%
4620     \bgroup
4621     \select@language@x{\bbl@main@language}%
4622     \bbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
4623     \egroup}
4624   \def\BabelFootnote#1#2#3#4{%
4625     \ifx\bbl@fn@footnote\undefined
4626       \let\bbl@fn@footnote\footnote
4627     \fi
4628     \ifx\bbl@fn@footnotetext\undefined
4629       \let\bbl@fn@footnotetext\footnotetext
4630     \fi
4631     \bbl@ifblank{#2}%
4632       {\def#1{\bbl@footnote{\@firstofone}{#3}{#4}}
4633        \@namedef{\bbl@stripslash#1text}%
4634          {\bbl@footnotetext{\@firstofone}{#3}{#4}}}%
4635       {\def#1{\bbl@exp{\bbl@footnote{\foreignlanguage{#2}}}{#3}{#4}}%
4636        \@namedef{\bbl@stripslash#1text}%
4637          {\bbl@exp{\bbl@footnotetext{\foreignlanguage{#2}}}{#3}{#4}}}%
4638   \fi
4639 <</Footnote changes>>
```

Now, the code.

```
4640 <*xetex>
4641 \def\BabelStringsDefault{unicode}
4642 \let\xebbl@stop\relax
4643 \AddBabelHook{xetex}{encodedcommands}{%
```

```

4644 \def\bb1@tempa{#1}%
4645 \ifx\bb1@tempa\@empty
4646 \XeTeXinputencoding"bytes"%
4647 \else
4648 \XeTeXinputencoding"#1"%
4649 \fi
4650 \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4651 \AddBabelHook{xetex}{stopcommands}{%
4652 \xebbl@stop
4653 \let\xebbl@stop\relax}
4654 \def\bb1@intraspace#1 #2 #3\@@{%
4655 \bb1@csarg\gdef{xeisp@\language}%
4656 {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4657 \def\bb1@intrapenalty#1\@@{%
4658 \bb1@csarg\gdef{xeipn@\language}%
4659 {\XeTeXlinebreakpenalty #1\relax}}
4660 \def\bb1@provide@intraspace{%
4661 \bb1@xin@{\bb1@cl{lnbrk}}{s}%
4662 \ifin@else\bb1@xin@{\bb1@cl{lnbrk}}{c}\fi
4663 \ifin@
4664 \bb1@ifunset{bb1@intsp@\language}{}%
4665 {\expandafter\ifx\csname bb1@intsp@\language\endcsname\@empty\else
4666 \ifx\bb1@KVP@intraspace\@nil
4667 \bb1@exp{%
4668 \\\bb1@intraspace\bb1@cl{intsp}\\\@@}%
4669 \fi
4670 \ifx\bb1@KVP@intrapenalty\@nil
4671 \bb1@intrapenalty0\@@
4672 \fi
4673 \fi
4674 \ifx\bb1@KVP@intraspace\@nil\else % We may override the ini
4675 \expandafter\bb1@intraspace\bb1@KVP@intraspace\@@
4676 \fi
4677 \ifx\bb1@KVP@intrapenalty\@nil\else
4678 \expandafter\bb1@intrapenalty\bb1@KVP@intrapenalty\@@
4679 \fi
4680 \bb1@exp{%
4681 \\\bb1@add\<extras\language>{%
4682 \XeTeXlinebreaklocale "\bb1@cl{tbc}"%
4683 \<bb1@xeisp@\language>%
4684 \<bb1@xeipn@\language>}%
4685 \\\bb1@tglobal\<extras\language>%
4686 \\\bb1@add\<noextras\language>{%
4687 \XeTeXlinebreaklocale "en"%
4688 \\\bb1@tglobal\<noextras\language>}%
4689 \ifx\bb1@ispace\@undefined
4690 \gdef\bb1@ispace{\bb1@cl{xeisp}}%
4691 \ifx\AtBeginDocument\@notprerr
4692 \expandafter\@secondoftwo % to execute right now
4693 \fi
4694 \AtBeginDocument{%
4695 \expandafter\bb1@add
4696 \csname selectfont \endcsname{\bb1@ispace}%
4697 \expandafter\bb1@tglobal\csname selectfont \endcsname}%
4698 \fi}%
4699 \fi}
4700 \ifx\DisableBabelHook\@undefined\endinput\fi
4701 \AddBabelHook{babel-fontspec}{afterextras}{\bb1@switchfont}
4702 \AddBabelHook{babel-fontspec}{beforestart}{\bb1@ckeckstdfonts}

```



```

4703 \DisableBabelHook{babel-fontspec}
4704 <<Font selection>>
4705 \input txtbabel.def
4706 </xetex>

```

## 13.2 Layout

*In progress.*

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

\bbl@startskip and \bbl@endskip are available to package authors. Thanks to the T<sub>E</sub>X 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 pdf<sub>te</sub>x and xet<sub>ex</sub>.

```

4707 <*texxet>
4708 \providecommand\bbl@provide@intraspace{}
4709 \bbl@trace{Redefinitions for bidi layout}
4710 \def\bbl@sspre@caption{%
4711   \bbl@exp{\everyhbox{\bbl@textdir\bbl@cs{wdir@\bbl@main@language}}}}
4712 \ifx\bbl@opt@layout\@nnil\endinput\fi % No layout
4713 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
4714 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
4715 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
4716   \def\@hangfrom#1{%
4717     \setbox\@tempboxa\hbox{#1}%
4718     \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
4719     \noindent\box\@tempboxa}
4720 \def\raggedright{%
4721   \let\@centercr
4722   \bbl@startskip\z@skip
4723   \@rightskip\@flushglue
4724   \bbl@endskip\@rightskip
4725   \parindent\z@
4726   \parfillskip\bbl@startskip}
4727 \def\raggedleft{%
4728   \let\@centercr
4729   \bbl@startskip\@flushglue
4730   \bbl@endskip\z@skip
4731   \parindent\z@
4732   \parfillskip\bbl@endskip}
4733 \fi
4734 \IfBabelLayout{lists}
4735   {\bbl@sreplace\list
4736     {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
4737     \def\bbl@listleftmargin{%
4738       \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
4739     \ifcase\bbl@engine
4740       \def\labelenumii{}\theenumii{}\pdfTeX doesn't reverse ()
4741       \def\p@enumiii{\p@enumii}\theenumii{}\fi
4742     \fi
4743     \bbl@sreplace\@verbatim
4744       {\leftskip\@totalleftmargin}%
4745       {\bbl@startskip\textwidth
4746         \advance\bbl@startskip-\linewidth}%
4747     \bbl@sreplace\@verbatim
4748       {\rightskip\z@skip}%
4749       {\bbl@endskip\z@skip}}%
4750   {}

```

```

4751 \IfBabelLayout{contents}
4752 {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
4753 \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
4754 {}
4755 \IfBabelLayout{columns}
4756 {\bbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbl@outputbox}%
4757 \def\bbl@outputbox#1{%
4758 \hb@xt@\textwidth{%
4759 \hskip\columnwidth
4760 \hfil
4761 {\normalcolor\vrule \@width\columnseprule}%
4762 \hfil
4763 \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
4764 \hskip-\textwidth
4765 \hb@xt@\columnwidth{\box\@outputbox \hss}%
4766 \hskip\columnsep
4767 \hskip\columnwidth}}}%
4768 {}
4769 <<Footnote changes>>
4770 \IfBabelLayout{footnotes}%
4771 {\BabelFootnote\footnote\language\language{}{}}%
4772 \BabelFootnote\localfootnote\language\language{}{}}%
4773 \BabelFootnote\mainfootnote{}{}}{}
4774 {}

```

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.

```

4775 \IfBabelLayout{counters}%
4776 {\let\bbl@latinarabic=\@arabic
4777 \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
4778 \let\bbl@asciroman=\@roman
4779 \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
4780 \let\bbl@asciiRoman=\@Roman
4781 \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}{}}
4782 </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, they 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`).

```

4783 (*luatex)
4784 \ifx\AddBabelHook\undefined % When plain.def, babel.sty starts
4785 \bbl@trace{Read language.dat}
4786 \ifx\bbl@readstream\undefined
4787   \csname newread\endcsname\bbl@readstream
4788 \fi
4789 \begingroup
4790   \toks@{}
4791   \count@ \z@ % 0=start, 1=0th, 2=normal
4792   \def\bbl@process@line#1#2 #3 #4 {%
4793     \ifx=#1%
4794       \bbl@process@synonym{#2}%
4795     \else
4796       \bbl@process@language{#1#2}{#3}{#4}%
4797     \fi
4798     \ignorespaces}
4799   \def\bbl@manylang{%
4800     \ifnum\bbl@last>\@ne
4801       \bbl@info{Non-standard hyphenation setup}%
4802     \fi
4803     \let\bbl@manylang\relax}
4804   \def\bbl@process@language#1#2#3{%
4805     \ifcase\count@
4806       \ifundefined{zth@#1}{\count@\tw@}{\count@\@ne}%
4807     \or
4808       \count@\tw@
4809     \fi
4810     \ifnum\count@=\tw@
4811       \expandafter\addlanguage\csname l@#1\endcsname
4812       \language\allocationnumber
4813       \chardef\bbl@last\allocationnumber
4814       \bbl@manylang
4815       \let\bbl@elt\relax
4816       \xdef\bbl@languages{%
4817         \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
4818     \fi
4819     \the\toks@
4820     \toks@{}}
4821   \def\bbl@process@synonym@aux#1#2{%
4822     \global\expandafter\chardef\csname l@#1\endcsname#2\relax
4823     \let\bbl@elt\relax
4824     \xdef\bbl@languages{%
4825       \bbl@languages\bbl@elt{#1}{#2}{\the\language}}%
4826   \def\bbl@process@synonym#1{%
4827     \ifcase\count@
4828       \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%

```

```

4829 \or
4830 \ifundefined{zth#1}{\bbl@process@synonym@aux{#1}{0}}{}%
4831 \else
4832 \bbl@process@synonym@aux{#1}{\the\bbl@last}%
4833 \fi}
4834 \ifx\bbl@languages\undefined % Just a (sensible?) guess
4835 \chardef\l@english\z@
4836 \chardef\l@USenglish\z@
4837 \chardef\bbl@last\z@
4838 \global\@namedef{bbl@hyphendata@0}{{hyphen.tex}}
4839 \gdef\bbl@languages{%
4840 \bbl@elt{english}{0}{hyphen.tex}}%
4841 \bbl@elt{USenglish}{0}{}%
4842 \else
4843 \global\let\bbl@languages@format\bbl@languages
4844 \def\bbl@elt#1#2#3#4{% Remove all except language 0
4845 \ifnum#2>\z@\else
4846 \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
4847 \fi}%
4848 \xdef\bbl@languages{\bbl@languages}%
4849 \fi
4850 \def\bbl@elt#1#2#3#4{\@namedef{zth#1}} % Define flags
4851 \bbl@languages
4852 \openin\bbl@readstream=language.dat
4853 \ifeof\bbl@readstream
4854 \bbl@warning{I couldn't find language.dat. No additional\\%
4855 patterns loaded. Reported}%
4856 \else
4857 \loop
4858 \endlinechar\m@ne
4859 \read\bbl@readstream to \bbl@line
4860 \endlinechar\^^M
4861 \if T\ifeof\bbl@readstream F\fi T\relax
4862 \ifx\bbl@line\empty\else
4863 \edef\bbl@line{\bbl@line\space\space\space}%
4864 \expandafter\bbl@process@line\bbl@line\relax
4865 \fi
4866 \repeat
4867 \fi
4868 \endgroup
4869 \bbl@trace{Macros for reading patterns files}
4870 \def\bbl@get@enc#1:#2:#3\@@{\def\bbl@hyph@enc{#2}}
4871 \ifx\babelcatcodetablenum\undefined
4872 \ifx\newcatcodetable\undefined
4873 \def\babelcatcodetablenum{5211}
4874 \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4875 \else
4876 \newcatcodetable\babelcatcodetablenum
4877 \newcatcodetable\bbl@pattcodes
4878 \fi
4879 \else
4880 \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4881 \fi
4882 \def\bbl@luapatterns#1#2{%
4883 \bbl@get@enc#1::\@@@
4884 \setbox\z@\hbox\bgroup
4885 \begingroup
4886 \savecatcodetable\babelcatcodetablenum\relax
4887 \initcatcodetable\bbl@pattcodes\relax

```

```

4888 \catcodetable\bbl@pattcodes\relax
4889 \catcode`\#=6 \catcode`\$=3 \catcode`\&=4 \catcode`\^=7
4890 \catcode`\_ =8 \catcode`\{=1 \catcode`\}=2 \catcode`\~=13
4891 \catcode`\@=11 \catcode`\^^I=10 \catcode`\^^J=12
4892 \catcode`\<=12 \catcode`\>=12 \catcode`\*=12 \catcode`\.=12
4893 \catcode`\-=12 \catcode`\/=12 \catcode`\[=12 \catcode`\]=12
4894 \catcode`\`=12 \catcode`\'=12 \catcode`\\"=12
4895 \input #1\relax
4896 \catcodetable\babelcatcodetablenum\relax
4897 \endgroup
4898 \def\bbl@tempa{#2}%
4899 \ifx\bbl@tempa@empty\else
4900 \input #2\relax
4901 \fi
4902 \egroup}%
4903 \def\bbl@patterns@lua#1{%
4904 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
4905 \csname l@#1\endcsname
4906 \edef\bbl@tempa{#1}%
4907 \else
4908 \csname l@#1:\f@encoding\endcsname
4909 \edef\bbl@tempa{#1:\f@encoding}%
4910 \fi\relax
4911 \@namedef{lu@texhyphen@loaded@the\language}{}% Temp
4912 \@ifundefined{bbl@hyphendata@the\language}%
4913 {\def\bbl@elt##1##2##3##4{%
4914 \ifnum##2=\csname l@bbl@tempa\endcsname % #2=spanish, dutch:OT1...
4915 \def\bbl@tempb{##3}%
4916 \ifx\bbl@tempb@empty\else % if not a synonymous
4917 \def\bbl@tempc{##3}{##4}%
4918 \fi
4919 \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
4920 \fi}%
4921 \bbl@languages
4922 \@ifundefined{bbl@hyphendata@the\language}%
4923 {\bbl@info{No hyphenation patterns were set for\%
4924 language '\bbl@tempa'. Reported}}%
4925 {\expandafter\expandafter\expandafter\bbl@luapatterns
4926 \csname bbl@hyphendata@the\language\endcsname}}}%
4927 \endinput\fi
4928 % Here ends \ifx\AddBabelHook\@undefined
4929 % A few lines are only read by hyphen.cfg
4930 \ifx\DisableBabelHook\@undefined
4931 \AddBabelHook{luatex}{everylanguage}{%
4932 \def\process@language##1##2##3{%
4933 \def\process@line####1####2 ####3 ####4 {}}}
4934 \AddBabelHook{luatex}{loadpatterns}{%
4935 \input #1\relax
4936 \expandafter\gdef\csname bbl@hyphendata@the\language\endcsname
4937 {#1}{}}
4938 \AddBabelHook{luatex}{loadexceptions}{%
4939 \input #1\relax
4940 \def\bbl@tempb##1##2{##1}{#1}%
4941 \expandafter\xdef\csname bbl@hyphendata@the\language\endcsname
4942 {\expandafter\expandafter\expandafter\bbl@tempb
4943 \csname bbl@hyphendata@the\language\endcsname}}
4944 \endinput\fi
4945 % Here stops reading code for hyphen.cfg
4946 % The following is read the 2nd time it's loaded

```

```

4947 \begingroup % TODO - to a lua file
4948 \catcode`\%=12
4949 \catcode`\'=12
4950 \catcode`\ "=12
4951 \catcode`\:=12
4952 \directlua{
4953   Babel = Babel or {}
4954   function Babel.bytes(line)
4955     return line:gsub(".",
4956       function (chr) return unicode.utf8.char(string.byte(chr)) end)
4957   end
4958   function Babel.begin_process_input()
4959     if luatexbase and luatexbase.add_to_callback then
4960       luatexbase.add_to_callback('process_input_buffer',
4961         Babel.bytes, 'Babel.bytes')
4962     else
4963       Babel.callback = callback.find('process_input_buffer')
4964       callback.register('process_input_buffer', Babel.bytes)
4965     end
4966   end
4967   function Babel.end_process_input ()
4968     if luatexbase and luatexbase.remove_from_callback then
4969       luatexbase.remove_from_callback('process_input_buffer', 'Babel.bytes')
4970     else
4971       callback.register('process_input_buffer', Babel.callback)
4972     end
4973   end
4974   function Babel.addpatterns(pp, lg)
4975     local lg = lang.new(lg)
4976     local pats = lang.patterns(lg) or ''
4977     lang.clear_patterns(lg)
4978     for p in pp:gmatch('[^%s]+') do
4979       ss = ''
4980       for i in string.utfcharacters(p:gsub('%d', '')) do
4981         ss = ss .. '%d?' .. i
4982       end
4983       ss = ss:gsub('^%%d%?%', '%%.') .. '%d?'
4984       ss = ss:gsub('%.%%d%?$', '%%.')
4985       pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
4986       if n == 0 then
4987         tex.sprint(
4988           [[\string\csname\space bbl@info\endcsname{New pattern: }]]
4989           .. p .. [[{}]])
4990         pats = pats .. ' ' .. p
4991       else
4992         tex.sprint(
4993           [[\string\csname\space bbl@info\endcsname{Renew pattern: }]]
4994           .. p .. [[{}]])
4995       end
4996     end
4997     lang.patterns(lg, pats)
4998   end
4999 }
5000 \endgroup
5001 \ifx\newattribute\@undefined\else
5002   \newattribute\bbl@attr@locale
5003   \directlua{ Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale'}
5004   \AddBabelHook{luatex}{beforeextras}{%
5005     \setattribute\bbl@attr@locale\localeid}

```

```

5006 \fi
5007 \def\BabelStringsDefault{unicode}
5008 \let\luabbl@stop\relax
5009 \AddBabelHook{luatex}{encodedcommands}{%
5010   \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
5011   \ifx\bbl@tempa\bbl@tempb\else
5012     \directlua{Babel.begin_process_input()}%
5013     \def\luabbl@stop{%
5014       \directlua{Babel.end_process_input()}}%
5015   \fi}%
5016 \AddBabelHook{luatex}{stopcommands}{%
5017   \luabbl@stop
5018   \let\luabbl@stop\relax}
5019 \AddBabelHook{luatex}{patterns}{%
5020   \@ifundefined{bbl@hyphendata@the\language}%
5021     {\def\bbl@elt##1##2##3##4{%
5022       \ifnum##2=\csname l@#2\endcsname % #2=spanish, dutch:OT1...
5023       \def\bbl@tempb{##3}%
5024       \ifx\bbl@tempb\@empty\else % if not a synonymous
5025         \def\bbl@tempc{##3}{##4}}%
5026       \fi
5027       \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
5028     \fi}%
5029   \bbl@languages
5030   \@ifundefined{bbl@hyphendata@the\language}%
5031     {\bbl@info{No hyphenation patterns were set for\%
5032       language '#2'. Reported}}%
5033     {\expandafter\expandafter\expandafter\bbl@luapatterns
5034       \csname bbl@hyphendata@the\language\endcsname}}}%
5035   \@ifundefined{bbl@patterns@}{}%
5036   \begingroup
5037     \bbl@xin@{,\number\language,}{,\bbl@pttnlist}%
5038     \ifin@else
5039       \ifx\bbl@patterns@\@empty\else
5040         \directlua{ Babel.addpatterns(
5041           [[\bbl@patterns@]], \number\language) }%
5042         \fi
5043         \@ifundefined{bbl@patterns@#1}%
5044         \@empty
5045         {\directlua{ Babel.addpatterns(
5046           [[\space\csname bbl@patterns@#1\endcsname]],
5047           \number\language) }}%
5048         \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%
5049       \fi
5050     \endgroup}%
5051   \bbl@exp{%
5052     \bbl@ifunset{bbl@prehc@the\language}{}%
5053     {\bbl@ifblank{\bbl@cs{prehc@the\language}}{}}%
5054     {\prehyphenchar=\bbl@c1{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.

```

5055 \@onlypreamble\babelpatterns
5056 \AtEndOfPackage{%
5057   \newcommand\babelpatterns[2][\@empty]{%
5058     \ifx\bbl@patterns@\relax
5059       \let\bbl@patterns@\@empty
5060     \fi

```

```

5061 \ifx\bbbl@pttnlist\@empty\else
5062 \bbbl@warning{%
5063   You must not intermingle \string\selectlanguage\space and\\%
5064   \string\babelpatterns\space or some patterns will not\\%
5065   be taken into account. Reported}%
5066 \fi
5067 \ifx\@empty#1%
5068 \protected@edef\bbbl@patterns@\bbbl@patterns@\space#2}%
5069 \else
5070 \edef\bbbl@tempb{\zap@space#1 \@empty}%
5071 \bbbl@for\bbbl@tempa\bbbl@tempb{%
5072   \bbbl@fixname\bbbl@tempa
5073   \bbbl@iflanguage\bbbl@tempa{%
5074     \bbbl@csarg\protected@edef{patterns@\bbbl@tempa}{%
5075       \ifundefined{bbbl@patterns@\bbbl@tempa}%
5076         \@empty
5077       {\csname bbl@patterns@\bbbl@tempa\endcsname\space}%
5078       #2}}}%
5079 \fi}}

```

### 13.4 Southeast Asian scripts

First, some general code for line breaking, used by `\babelposthyphenation`.  
 Replace regular (ie, implicit) discretionaries by spaceskips, based on the previous glyph (which I think makes sense, because the hyphen and the previous char go always together). Other discretionaries are not touched. See Unicode UAX 14.

```

5080% TODO - to a lua file
5081 \directlua{
5082   Babel = Babel or {}
5083   Babel.linebreaking = Babel.linebreaking or {}
5084   Babel.linebreaking.before = {}
5085   Babel.linebreaking.after = {}
5086   Babel.locale = {} % Free to use, indexed by \localeid
5087   function Babel.linebreaking.add_before(func)
5088     tex.print([[ \noexpand\csname bbl@luahyphenate\endcsname]])
5089     table.insert(Babel.linebreaking.before, func)
5090   end
5091   function Babel.linebreaking.add_after(func)
5092     tex.print([[ \noexpand\csname bbl@luahyphenate\endcsname]])
5093     table.insert(Babel.linebreaking.after, func)
5094   end
5095 }
5096 \def\bbbl@intraspace#1 #2 #3\@{#%
5097 \directlua{
5098   Babel = Babel or {}
5099   Babel.intraspaces = Babel.intraspaces or {}
5100   Babel.intraspaces['\csname bbl@sbc@language\endcsname'] = %
5101     {b = #1, p = #2, m = #3}
5102   Babel.locale_props[\the\localeid].intraspace = %
5103     {b = #1, p = #2, m = #3}
5104 }}
5105 \def\bbbl@intrapenalty#1\@{#%
5106 \directlua{
5107   Babel = Babel or {}
5108   Babel.intrapenalties = Babel.intrapenalties or {}
5109   Babel.intrapenalties['\csname bbl@sbc@language\endcsname'] = #1
5110   Babel.locale_props[\the\localeid].intrapenalty = #1
5111 }}

```



```

5112 \begingroup
5113 \catcode`\%=12
5114 \catcode`\^=14
5115 \catcode`\'=12
5116 \catcode`\~=12
5117 \gdef\bbl@seaintraspace{^
5118   \let\bbl@seaintraspace\relax
5119   \directlua{
5120     Babel = Babel or {}
5121     Babel.sea_enabled = true
5122     Babel.sea_ranges = Babel.sea_ranges or {}
5123     function Babel.set_chranges (script, chrng)
5124       local c = 0
5125       for s, e in string.gmatch(chrng..' ', '(.-%.%.(-)%s') do
5126         Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
5127         c = c + 1
5128       end
5129     end
5130     function Babel.sea_disc_to_space (head)
5131       local sea_ranges = Babel.sea_ranges
5132       local last_char = nil
5133       local quad = 655360      ^% 10 pt = 655360 = 10 * 65536
5134       for item in node.traverse(head) do
5135         local i = item.id
5136         if i == node.id'glyph' then
5137           last_char = item
5138         elseif i == 7 and item.subtype == 3 and last_char
5139           and last_char.char > 0x0C99 then
5140           quad = font.getfont(last_char.font).size
5141           for lg, rg in pairs(sea_ranges) do
5142             if last_char.char > rg[1] and last_char.char < rg[2] then
5143               lg = lg:sub(1, 4)  ^% Remove trailing number of, eg, Cyr11
5144               local intraspace = Babel.intraspaces[lg]
5145               local intrapenalty = Babel.intrapenalties[lg]
5146               local n
5147               if intrapenalty ~= 0 then
5148                 n = node.new(14, 0)      ^% penalty
5149                 n.penalty = intrapenalty
5150                 node.insert_before(head, item, n)
5151               end
5152               n = node.new(12, 13)      ^% (glue, spaceskip)
5153               node.setglue(n, intraspace.b * quad,
5154                 intraspace.p * quad,
5155                 intraspace.m * quad)
5156               node.insert_before(head, item, n)
5157               node.remove(head, item)
5158             end
5159           end
5160         end
5161       end
5162     end
5163   }^^
5164   \bbl@luahyphenate}
5165 \catcode`\%=14
5166 \gdef\bbl@cjkintraspacespace{%
5167   \let\bbl@cjkintraspacespace\relax
5168   \directlua{
5169     Babel = Babel or {}
5170     require('babel-data-cjk.lua')

```

```

5171 Babel.cjk_enabled = true
5172 function Babel.cjk_linebreak(head)
5173     local GLYPH = node.id'glyph'
5174     local last_char = nil
5175     local quad = 655360      % 10 pt = 655360 = 10 * 65536
5176     local last_class = nil
5177     local last_lang = nil
5178
5179     for item in node.traverse(head) do
5180         if item.id == GLYPH then
5181
5182             local lang = item.lang
5183
5184             local LOCALE = node.get_attribute(item,
5185                 luatexbase.registernumber'bbl@attr@locale')
5186             local props = Babel.locale_props[LOCALE]
5187
5188             local class = Babel.cjk_class[item.char].c
5189
5190             if class == 'cp' then class = 'cl' end % ]] as CL
5191             if class == 'id' then class = 'I' end
5192
5193             local br = 0
5194             if class and last_class and Babel.cjk_breaks[last_class][class] then
5195                 br = Babel.cjk_breaks[last_class][class]
5196             end
5197
5198             if br == 1 and props.linebreak == 'c' and
5199                 lang ~= \the\l@nohyphenation\space and
5200                 last_lang ~= \the\l@nohyphenation then
5201                 local intrapenalty = props.intrapenalty
5202                 if intrapenalty ~= 0 then
5203                     local n = node.new(14, 0)      % penalty
5204                     n.penalty = intrapenalty
5205                     node.insert_before(head, item, n)
5206                 end
5207                 local intraspace = props.intraspace
5208                 local n = node.new(12, 13)          % (glue, spaceskip)
5209                 node.setglue(n, intraspace.b * quad,
5210                     intraspace.p * quad,
5211                     intraspace.m * quad)
5212                 node.insert_before(head, item, n)
5213             end
5214
5215             if font.getfont(item.font) then
5216                 quad = font.getfont(item.font).size
5217             end
5218             last_class = class
5219             last_lang = lang
5220             else % if penalty, glue or anything else
5221                 last_class = nil
5222             end
5223         end
5224         lang.hyphenate(head)
5225     end
5226 }%
5227 \bbl@luahyphenate}
5228 \gdef\bbl@luahyphenate{%
5229 \let\bbl@luahyphenate\relax

```

```

5230 \directlua{
5231   luatexbase.add_to_callback('hyphenate',
5232   function (head, tail)
5233     if Babel.linebreaking.before then
5234       for k, func in ipairs(Babel.linebreaking.before) do
5235         func(head)
5236       end
5237     end
5238     if Babel.cjk_enabled then
5239       Babel.cjk_linebreak(head)
5240     end
5241     lang.hyphenate(head)
5242     if Babel.linebreaking.after then
5243       for k, func in ipairs(Babel.linebreaking.after) do
5244         func(head)
5245       end
5246     end
5247     if Babel.sea_enabled then
5248       Babel.sea_disc_to_space(head)
5249     end
5250   end,
5251   'Babel.hyphenate')
5252 }
5253 }
5254 \endgroup
5255 \def\bbl@provide@intraspace{%
5256   \bbl@ifunset{\bbl@intsp@language}{}%
5257   {\expandafter\ifx\cename\bbl@intsp@language\endcsname\@empty\else
5258     \bbl@xin@{\bbl@cl{lnbrk}}{c}%
5259     \ifin@           % cjk
5260     \bbl@cjk@intraspace
5261     \directlua{
5262       Babel = Babel or {}
5263       Babel.locale_props = Babel.locale_props or {}
5264       Babel.locale_props[\the\localeid].linebreak = 'c'
5265     }%
5266     \bbl@exp{\bbl@intraspace\bbl@cl{intsp}}{\@}%
5267     \ifx\bbl@KVP@intrapenalty\@nil
5268       \bbl@intrapenalty0\@@
5269     \fi
5270   \else           % sea
5271     \bbl@sea@intraspace
5272     \bbl@exp{\bbl@intraspace\bbl@cl{intsp}}{\@}%
5273     \directlua{
5274       Babel = Babel or {}
5275       Babel.sea_ranges = Babel.sea_ranges or {}
5276       Babel.set_chranges('\bbl@cl{sbcpr}',
5277         '\bbl@cl{chrng}')
5278     }%
5279     \ifx\bbl@KVP@intrapenalty\@nil
5280       \bbl@intrapenalty0\@@
5281     \fi
5282   \fi
5283 \fi
5284 \ifx\bbl@KVP@intrapenalty\@nil\else
5285   \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@@
5286 \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.

```
5287 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
5288 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@cckstdfont}
5289 \DisableBabelHook{babel-fontspec}
5290 <<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 from 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.

```
5291% TODO - to a lua file
5292 \directlua{
5293 Babel.script_blocks = {
5294   ['dflt'] = {},
5295   ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
5296              {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
5297   ['Armn'] = {{0x0530, 0x058F}},
5298   ['Beng'] = {{0x0980, 0x09FF}},
5299   ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0xABBF}},
5300   ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
5301   ['Cyr1'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
5302              {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
5303   ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
5304   ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF},
5305              {0xAB00, 0xAB2F}},
5306   ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
5307   % Don't follow strictly Unicode, which places some Coptic letters in
5308   % the 'Greek and Coptic' block
5309   ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
5310   ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
5311              {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
5312              {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
5313              {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
5314              {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
5315              {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
5316   ['Hebr'] = {{0x0590, 0x05FF}},
5317   ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
5318              {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
5319   ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
5320   ['Knda'] = {{0x0C80, 0x0CFF}},
5321   ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
5322              {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
5323              {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
5324   ['Lao'] = {{0x0E80, 0x0EFF}},
```

```

5325 ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
5326             {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
5327             {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
5328 ['Mahj'] = {{0x11150, 0x1117F}},
5329 ['Mlym'] = {{0x0D00, 0x0D7F}},
5330 ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
5331 ['Orya'] = {{0x0B00, 0x0B7F}},
5332 ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
5333 ['Syrn'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
5334 ['Taml'] = {{0x0B80, 0x0BFF}},
5335 ['Telu'] = {{0x0C00, 0x0C7F}},
5336 ['Tfng'] = {{0x2D30, 0x2D7F}},
5337 ['Thai'] = {{0x0E00, 0x0E7F}},
5338 ['Tibt'] = {{0x0F00, 0x0FFF}},
5339 ['Vaii'] = {{0xA500, 0xA63F}},
5340 ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
5341 }
5342
5343 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
5344 Babel.script_blocks.Hant = Babel.script_blocks.Hans
5345 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
5346
5347 function Babel.locale_map(head)
5348   if not Babel.locale_mapped then return head end
5349
5350   local LOCALE = luatexbase.registernumber'bbl@attr@locale'
5351   local GLYPH = node.id('glyph')
5352   local inmath = false
5353   local toloc_save
5354   for item in node.traverse(head) do
5355     local toloc
5356     if not inmath and item.id == GLYPH then
5357       % Optimization: build a table with the chars found
5358       if Babel.chr_to_loc[item.char] then
5359         toloc = Babel.chr_to_loc[item.char]
5360       else
5361         for lc, maps in pairs(Babel.loc_to_scr) do
5362           for _, rg in pairs(maps) do
5363             if item.char >= rg[1] and item.char <= rg[2] then
5364               Babel.chr_to_loc[item.char] = lc
5365               toloc = lc
5366               break
5367             end
5368           end
5369         end
5370       end
5371       % Now, take action, but treat composite chars in a different
5372       % fashion, because they 'inherit' the previous locale. Not yet
5373       % optimized.
5374       if not toloc and
5375         (item.char >= 0x0300 and item.char <= 0x036F) or
5376         (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
5377         (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
5378         toloc = toloc_save
5379       end
5380       if toloc and toloc > -1 then
5381         if Babel.locale_props[toloc].lg then
5382           item.lang = Babel.locale_props[toloc].lg
5383           node.set_attribute(item, LOCALE, toloc)

```

```

5384         end
5385         if Babel.locale_props[toloc]['/'..item.font] then
5386             item.font = Babel.locale_props[toloc]['/'..item.font]
5387         end
5388         toloc_save = toloc
5389     end
5390     elseif not inmath and item.id == 7 then
5391         item.replace = item.replace and Babel.locale_map(item.replace)
5392         item.pre      = item.pre and Babel.locale_map(item.pre)
5393         item.post     = item.post and Babel.locale_map(item.post)
5394     elseif item.id == node.id'math' then
5395         inmath = (item.subtype == 0)
5396     end
5397 end
5398 return head
5399 end
5400 }

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

5401 \newcommand\babelcharproperty[1]{%
5402   \count@=#1\relax
5403   \ifvmode
5404     \expandafter\bbl@chprop
5405   \else
5406     \bbl@error{\string\babelcharproperty\space can be used only in\\%
5407               vertical mode (preamble or between paragraphs)}%
5408               {See the manual for futher info}%
5409   \fi}
5410 \newcommand\bbl@chprop[3][\the\count@]{%
5411   \@tempcnta=#1\relax
5412   \bbl@ifunset{\bbl@chprop@#2}%
5413   {\bbl@error{No property named '#2'. Allowed values are\\%
5414             direction (bc), mirror (bmg), and linebreak (lb)}%
5415             {See the manual for futher info}}%
5416   }%
5417   \loop
5418     \bbl@cs{chprop@#2}{#3}%
5419     \ifnum\count@<\@tempcnta
5420       \advance\count@\@ne
5421     \repeat}
5422 \def\bbl@chprop@direction#1{%
5423   \directlua{
5424     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5425     Babel.characters[\the\count@]['d'] = '#1'
5426   }}
5427 \let\bbl@chprop@bc\bbl@chprop@direction
5428 \def\bbl@chprop@mirror#1{%
5429   \directlua{
5430     Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5431     Babel.characters[\the\count@]['m'] = '\number#1'
5432   }}
5433 \let\bbl@chprop@bmg\bbl@chprop@mirror
5434 \def\bbl@chprop@linebreak#1{%
5435   \directlua{
5436     Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
5437     Babel.cjk_characters[\the\count@]['c'] = '#1'
5438   }}
5439 \let\bbl@chprop@lb\bbl@chprop@linebreak

```

```

5440 \def\bbl@chprop@locale#1{%
5441   \directlua{
5442     Babel.chr_to_loc = Babel.chr_to_loc or {}
5443     Babel.chr_to_loc[\the\count@] =
5444       \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@#1}}\space
5445   }}

```

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.

```

5446 \begingroup % TODO - to a lua file
5447 \catcode`\~=12
5448 \catcode`\#=12
5449 \catcode`\%=12
5450 \catcode`\&=14
5451 \directlua{
5452   Babel.linebreaking.replacements = {}
5453   Babel.linebreaking.replacements[0] = {}  %% pre
5454   Babel.linebreaking.replacements[1] = {}  %% post
5455
5456   %% Discretionaries contain strings as nodes
5457   function Babel.str_to_nodes(fn, matches, base)
5458     local n, head, last
5459     if fn == nil then return nil end
5460     for s in string.utfvalues(fn(matches)) do
5461       if base.id == 7 then
5462         base = base.replace
5463       end
5464       n = node.copy(base)
5465       n.char = s
5466       if not head then
5467         head = n
5468       else
5469         last.next = n
5470       end
5471       last = n
5472     end
5473     return head
5474   end
5475
5476   Babel.fetch_subtext = {}
5477
5478   %% Merging both functions doesn't seem feasible, because there are too
5479   %% many differences.
5480   Babel.fetch_subtext[0] = function(head)
5481     local word_string = ''
5482     local word_nodes = {}
5483     local lang
5484     local item = head

```

```

5485     local inmath = false
5486
5487     while item do
5488
5489         if item.id == 11 then
5490             inmath = (item.subtype == 0)
5491         end
5492
5493         if inmath then
5494             %% pass
5495
5496         elseif item.id == 29 then
5497             local locale = node.get_attribute(item, Babel.attr_locale)
5498
5499             if lang == locale or lang == nil then
5500                 lang = lang or locale
5501                 word_string = word_string .. unicode.utf8.char(item.char)
5502                 word_nodes[#word_nodes+1] = item
5503             else
5504                 break
5505             end
5506
5507         elseif item.id == 12 and item.subtype == 13 then
5508             word_string = word_string .. ' '
5509             word_nodes[#word_nodes+1] = item
5510
5511             %% Ignore leading unrecognized nodes, too.
5512             elseif word_string ~= '' then
5513                 word_string = word_string .. Babel.us_char
5514                 word_nodes[#word_nodes+1] = item %% Will be ignored
5515             end
5516
5517             item = item.next
5518         end
5519
5520         %% Here and above we remove some trailing chars but not the
5521         %% corresponding nodes. But they aren't accessed.
5522         if word_string:sub(-1) == ' ' then
5523             word_string = word_string:sub(1,-2)
5524         end
5525         word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
5526         return word_string, word_nodes, item, lang
5527     end
5528
5529     Babel.fetch_subtext[1] = function(head)
5530         local word_string = ''
5531         local word_nodes = {}
5532         local lang
5533         local item = head
5534         local inmath = false
5535
5536         while item do
5537
5538             if item.id == 11 then
5539                 inmath = (item.subtype == 0)
5540             end
5541
5542             if inmath then
5543                 %% pass

```



```

5544
5545     elseif item.id == 29 then
5546         if item.lang == lang or lang == nil then
5547             if (item.char ~= 124) and (item.char ~= 61) then && not =, not |
5548                 lang = lang or item.lang
5549                 word_string = word_string .. unicode.utf8.char(item.char)
5550                 word_nodes[#word_nodes+1] = item
5551             end
5552         else
5553             break
5554         end
5555
5556     elseif item.id == 7 and item.subtype == 2 then
5557         word_string = word_string .. '='
5558         word_nodes[#word_nodes+1] = item
5559
5560     elseif item.id == 7 and item.subtype == 3 then
5561         word_string = word_string .. '|'
5562         word_nodes[#word_nodes+1] = item
5563
5564     && (1) Go to next word if nothing was found, and (2) implicitly
5565     && remove leading USs.
5566     elseif word_string == '' then
5567         && pass
5568
5569     && This is the responsible for splitting by words.
5570     elseif (item.id == 12 and item.subtype == 13) then
5571         break
5572
5573     else
5574         word_string = word_string .. Babel.us_char
5575         word_nodes[#word_nodes+1] = item && Will be ignored
5576     end
5577
5578     item = item.next
5579 end
5580
5581 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
5582 return word_string, word_nodes, item, lang
5583 end
5584
5585 function Babel.pre_hyphenate_replace(head)
5586     Babel.hyphenate_replace(head, 0)
5587 end
5588
5589 function Babel.post_hyphenate_replace(head)
5590     Babel.hyphenate_replace(head, 1)
5591 end
5592
5593 function Babel.debug_hyph(w, wn, sc, first, last, last_match)
5594     local ss = ''
5595     for pp = 1, 40 do
5596         if wn[pp] then
5597             if wn[pp].id == 29 then
5598                 ss = ss .. unicode.utf8.char(wn[pp].char)
5599             else
5600                 ss = ss .. '{' .. wn[pp].id .. '}'
5601             end
5602         end

```

```

5603     end
5604     print('nod', ss)
5605     print('lst_m',
5606           string.rep(' ', unicode.utf8.len(
5607             string.sub(w, 1, last_match))-1) .. '>')
5608     print('str', w)
5609     print('sc', string.rep(' ', sc-1) .. '^')
5610     if first == last then
5611       print('f=l', string.rep(' ', first-1) .. '!!')
5612     else
5613       print('f/l', string.rep(' ', first-1) .. '[' ..
5614             string.rep(' ', last-first-1) .. ']')
5615     end
5616   end
5617
5618   Babel.us_char = string.char(31)
5619
5620   function Babel.hyphenate_replace(head, mode)
5621     local u = unicode.utf8
5622     local lbkr = Babel.linebreaking.replacements[mode]
5623
5624     local word_head = head
5625
5626     while true do  %% for each subtext block
5627
5628       local w, w_nodes, nw, lang = Babel.fetch_subtext[mode](word_head)
5629
5630       if Babel.debug then
5631         print()
5632         print((mode == 0) and '@@@<' or '@@@>', w)
5633       end
5634
5635       if nw == nil and w == '' then break end
5636
5637       if not lang then goto next end
5638       if not lbkr[lang] then goto next end
5639
5640       %% For each saved (pre|post)hyphenation. TODO. Reconsider how
5641       %% loops are nested.
5642       for k=1, #lbkr[lang] do
5643         local p = lbkr[lang][k].pattern
5644         local r = lbkr[lang][k].replace
5645
5646         if Babel.debug then
5647           print('*****', p, mode)
5648         end
5649
5650         %% This variable is set in some cases below to the first *byte*
5651         %% after the match, either as found by u.match (faster) or the
5652         %% computed position based on sc if w has changed.
5653         local last_match = 0
5654
5655         %% For every match.
5656         while true do
5657           if Babel.debug then
5658             print('====')
5659           end
5660           local new  %% used when inserting and removing nodes
5661           local refetch = false

```

```

5662
5663     local matches = { u.match(w, p, last_match) }
5664     if #matches < 2 then break end
5665
5666     %% Get and remove empty captures (with ())'s, which return a
5667     %% number with the position), and keep actual captures
5668     %% (from (...)), if any, in matches.
5669     local first = table.remove(matches, 1)
5670     local last  = table.remove(matches, #matches)
5671     %% Non re-fetched substrings may contain \31, which separates
5672     %% subsubstrings.
5673     if string.find(w:sub(first, last-1), Babel.us_char) then break end
5674
5675     local save_last = last %% with A()BC()D, points to D
5676
5677     %% Fix offsets, from bytes to unicode. Explained above.
5678     first = u.len(w:sub(1, first-1)) + 1
5679     last  = u.len(w:sub(1, last-1)) %% now last points to C
5680
5681     %% This loop stores in n small table the nodes
5682     %% corresponding to the pattern. Used by 'data' to provide a
5683     %% predictable behavior with 'insert' (now w_nodes is modified on
5684     %% the fly), and also access to 'remove'd nodes.
5685     local sc = first-1          %% Used below, too
5686     local data_nodes = {}
5687
5688     for q = 1, last-first+1 do
5689         data_nodes[q] = w_nodes[sc+q]
5690     end
5691
5692     %% This loop traverses the matched substring and takes the
5693     %% corresponding action stored in the replacement list.
5694     %% sc = the position in substr nodes / string
5695     %% rc = the replacement table index
5696     local rc = 0
5697
5698     while rc < last-first+1 do %% for each replacement
5699         if Babel.debug then
5700             print('.....', rc + 1)
5701         end
5702         sc = sc + 1
5703         rc = rc + 1
5704
5705         if Babel.debug then
5706             Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
5707             local ss = ''
5708             for itt in node.traverse(head) do
5709                 if itt.id == 29 then
5710                     ss = ss .. unicode.utf8.char(itt.char)
5711                 else
5712                     ss = ss .. '{' .. itt.id .. '}'
5713                 end
5714             end
5715             print('*****', ss)
5716         end
5717
5718         local crep = r[rc]
5719         local item = w_nodes[sc]
5720

```

```

5721     local item_base = item
5722     local placeholder = Babel.us_char
5723     local d
5724
5725     if crep and crep.data then
5726         item_base = data_nodes[crep.data]
5727     end
5728
5729     if crep and next(crep) == nil then &% = {}
5730         last_match = save_last    &% Optimization
5731         goto next
5732
5733     elseif crep == nil or crep.remove then
5734         node.remove(head, item)
5735         table.remove(w_nodes, sc)
5736         w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
5737         sc = sc - 1    &% Nothing has been inserted.
5738         last_match = utf8.offset(w, sc+1)
5739         goto next
5740
5741     elseif crep and crep.string then
5742         local str = crep.string(matches)
5743         if str == '' then    &% Gather with nil
5744             node.remove(head, item)
5745             table.remove(w_nodes, sc)
5746             w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
5747             sc = sc - 1    &% Nothing has been inserted.
5748         else
5749             local loop_first = true
5750             for s in string.utfvalues(str) do
5751                 d = node.copy(item_base)
5752                 d.char = s
5753                 if loop_first then
5754                     loop_first = false
5755                     head, new = node.insert_before(head, item, d)
5756                     if sc == 1 then
5757                         word_head = head
5758                     end
5759                     w_nodes[sc] = d
5760                     w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc+1)
5761                 else
5762                     sc = sc + 1
5763                     head, new = node.insert_before(head, item, d)
5764                     table.insert(w_nodes, sc, new)
5765                     w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc)
5766                 end
5767                 if Babel.debug then
5768                     print('.....', 'str')
5769                     Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
5770                 end
5771             end    &% for
5772             node.remove(head, item)
5773         end    &% if ''
5774         last_match = utf8.offset(w, sc+1)
5775         goto next
5776
5777     elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
5778         d = node.new(7, 0)    &% (disc, discretionary)
5779         d.pre    = Babel.str_to_nodes(crep.pre, matches, item_base)

```

```

5780         d.post      = Babel.str_to_nodes(crep.post, matches, item_base)
5781         d.replace    = Babel.str_to_nodes(crep.no, matches, item_base)
5782         d.attr       = item_base.attr
5783         if crep.pre == nil then  &% TeXbook p96
5784             d.penalty = crep.penalty or tex.hyphenpenalty
5785         else
5786             d.penalty = crep.penalty or tex.exhyphenpenalty
5787         end
5788         placeholder = '|'
5789         head, new = node.insert_before(head, item, d)
5790
5791     elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
5792         &% ERROR
5793
5794     elseif crep and crep.penalty then
5795         d = node.new(14, 0)  &% (penalty, userpenalty)
5796         d.attr = item_base.attr
5797         d.penalty = crep.penalty
5798         head, new = node.insert_before(head, item, d)
5799
5800     elseif crep and crep.space then
5801         &% 655360 = 10 pt = 10 * 65536 sp
5802         d = node.new(12, 13)  &% (glue, spaceskip)
5803         local quad = font.getfont(item_base.font).size or 655360
5804         node.setglue(d, crep.space[1] * quad,
5805                       crep.space[2] * quad,
5806                       crep.space[3] * quad)
5807         if mode == 0 then
5808             placeholder = ' '
5809         end
5810         head, new = node.insert_before(head, item, d)
5811
5812     elseif crep and crep.spacefactor then
5813         d = node.new(12, 13)  &% (glue, spaceskip)
5814         local base_font = font.getfont(item_base.font)
5815         node.setglue(d,
5816                     crep.spacefactor[1] * base_font.parameters['space'],
5817                     crep.spacefactor[2] * base_font.parameters['space_stretch'],
5818                     crep.spacefactor[3] * base_font.parameters['space_shrink'])
5819         if mode == 0 then
5820             placeholder = ' '
5821         end
5822         head, new = node.insert_before(head, item, d)
5823
5824     elseif mode == 0 and crep and crep.space then
5825         &% ERROR
5826
5827     end  &% ie replacement cases
5828
5829     &% Shared by disc, space and penalty.
5830     if sc == 1 then
5831         word_head = head
5832     end
5833     if crep.insert then
5834         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc)
5835         table.insert(w_nodes, sc, new)
5836         last = last + 1
5837     else
5838         w_nodes[sc] = d

```

```

5839         node.remove(head, item)
5840         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1)
5841     end
5842
5843     last_match = utf8.offset(w, sc+1)
5844
5845     ::next::
5846
5847     end &% for each replacement
5848
5849     if Babel.debug then
5850         print('.....', '/')
5851         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
5852     end
5853
5854     end &% for match
5855
5856     end &% for patterns
5857
5858     ::next::
5859     word_head = nw
5860 end &% for substring
5861 return head
5862 end
5863
5864 &% This table stores capture maps, numbered consecutively
5865 Babel.capture_maps = {}
5866
5867 &% The following functions belong to the next macro
5868 function Babel.capture_func(key, cap)
5869     local ret = "[" .. cap:gsub('{{([0-9])}}', "].m[%1]..[" .. "]"
5870     local cnt
5871     local u = unicode.utf8
5872     ret, cnt = ret:gsub('{{([0-9])|([^\]]+)|(.-)}}', Babel.capture_func_map)
5873     if cnt == 0 then
5874         ret = u.gsub(ret, '{{(%x%x%x%x+)}}',
5875             function (n)
5876                 return u.char(tonumber(n, 16))
5877             end)
5878     end
5879     ret = ret:gsub("%[%[%]]%.", '')
5880     ret = ret:gsub("%.%[%[%]]%", '')
5881     return key .. [[=function(m) return ]] .. ret .. [[ end]]
5882 end
5883
5884 function Babel.capt_map(from, mapno)
5885     return Babel.capture_maps[mapno][from] or from
5886 end
5887
5888 &% Handle the {n|abc|ABC} syntax in captures
5889 function Babel.capture_func_map(capno, from, to)
5890     local u = unicode.utf8
5891     from = u.gsub(from, '{{(%x%x%x%x+)}}',
5892         function (n)
5893             return u.char(tonumber(n, 16))
5894         end)
5895     to = u.gsub(to, '{{(%x%x%x%x+)}}',
5896         function (n)
5897             return u.char(tonumber(n, 16))

```

```

5898         end)
5899     local froms = {}
5900     for s in string.utfcharacters(from) do
5901         table.insert(froms, s)
5902     end
5903     local cnt = 1
5904     table.insert(Babel.capture_maps, {})
5905     local mlen = table.getn(Babel.capture_maps)
5906     for s in string.utfcharacters(to) do
5907         Babel.capture_maps[mlen][froms[cnt]] = s
5908         cnt = cnt + 1
5909     end
5910     return "]]..Babel.capt_map(m[" .. capno .. "], " ..
5911         (mlen) .. ").. " .. "["
5912 end
5913 }

```

Now the  $\TeX$  high level interface, which requires the function defined above for converting strings to functions returning a string. These functions handle the  $\{n\}$  syntax. For example,  $\text{pre}=\{1\}\{1\}$ - 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 is 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).

```

5914 \catcode`\#=6
5915 \gdef\babelposthyphenation#1#2#3{&%
5916   \bbl@activateposthyphen
5917   \begingroup
5918     \def\babeltempa{\bbl@add@list\babeltempb}&%
5919     \let\babeltempb\@empty
5920     \def\bbl@tempa{#3}&% TODO. Ugly trick to preserve {}:
5921     \bbl@replace\bbl@tempa{,}{ ,}&%
5922     \expandafter\bbl@foreach\expandafter{\bbl@tempa}{&%
5923       \bbl@ifsamestring{##1}{remove}&%
5924       {\bbl@add@list\babeltempb{nil}}&%
5925       {\directlua{
5926         local rep = {[#1]=]
5927         rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')
5928         rep = rep:gsub('^%s*(insert)%s*', 'insert = true, ')
5929         rep = rep:gsub(' (no)%s*=%s*([^\s,]*)', Babel.capture_func)
5930         rep = rep:gsub(' (pre)%s*=%s*([^\s,]*)', Babel.capture_func)
5931         rep = rep:gsub(' (post)%s*=%s*([^\s,]*)', Babel.capture_func)
5932         rep = rep:gsub('(string)%s*=%s*([^\s,]*)', Babel.capture_func)
5933         tex.print([[\\string\babeltempa{}}] .. rep .. [[]]])
5934       }}}&%
5935   \directlua{
5936     local lbkr = Babel.linebreaking.replacements[1]
5937     local u = unicode.utf8
5938     local id = \the\csname l@#1\endcsname
5939     &% Convert pattern:
5940     local patt = string.gsub([=[#2]=], '%s', '')
5941     if not u.find(patt, '()', nil, true) then
5942       patt = '()' .. patt .. '()'
5943     end
5944     patt = string.gsub(patt, '%(%)%^\', '^()')
5945     patt = string.gsub(patt, '%$%(%)', '()$')
5946     patt = u.gsub(patt, '{(.)}',

```

```

5947         function (n)
5948             return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
5949         end)
5950     patt = u.gsub(patt, '{(%x%x%x%x+)}',
5951         function (n)
5952             return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%1')
5953         end)
5954     lbkr[id] = lbkr[id] or {}
5955     table.insert(lbkr[id], { pattern = patt, replace = { \babeltempb } })
5956 }&%
5957 \endgroup}
5958 % TODO. Copypaste pattern.
5959 \gdef\babelprehyphenation#1#2#3{&%
5960     \bbl@activateprehyphen
5961     \beginingroup
5962     \def\babeltempa{\bbl@add@list\babeltempb}&%
5963     \let\babeltempb\@empty
5964     \def\bbl@tempa{#3}&% TODO. Ugly trick to preserve {}:
5965     \bbl@replace\bbl@tempa{,}{ ,}&%
5966     \expandafter\bbl@foreach\expandafter{\bbl@tempa}{&%
5967         \bbl@ifsamestring{##1}{remove}&%
5968         {\bbl@add@list\babeltempb{nil}}&%
5969         {\directlua{
5970             local rep = [#1]=]
5971             rep = rep.gsub('^s*(remove)s*$', 'remove = true')
5972             rep = rep.gsub('^s*(insert)s*', 'insert = true, ')
5973             rep = rep.gsub('(string)s*=%s*([^\s,]*)', Babel.capture_func)
5974             rep = rep.gsub(' (space)s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
5975                 'space = {' .. '%2, %3, %4' .. '}')
5976             rep = rep.gsub(' (spacefactor)s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
5977                 'spacefactor = {' .. '%2, %3, %4' .. '}')
5978             tex.print([[\string\babeltempa{}}] .. rep .. [[]]])
5979         }}&%
5980     \directlua{
5981         local lbkr = Babel.linebreaking.replacements[0]
5982         local u = unicode.utf8
5983         local id = \the\csname bbl@id@#1\endcsname
5984         &% Convert pattern:
5985         local patt = string.gsub([=#2]=], '%s', '')
5986         local patt = string.gsub(patt, '|', ' ')
5987         if not u.find(patt, '()', nil, true) then
5988             patt = '()' .. patt .. '()'
5989         end
5990         &% patt = string.gsub(patt, '%(%)^', '^()')
5991         &% patt = string.gsub(patt, '([^\%])%%$', '%1()$')
5992         patt = u.gsub(patt, '{(.)}',
5993             function (n)
5994                 return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
5995             end)
5996         patt = u.gsub(patt, '{(%x%x%x%x+)}',
5997             function (n)
5998                 return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%1')
5999             end)
6000         lbkr[id] = lbkr[id] or {}
6001         table.insert(lbkr[id], { pattern = patt, replace = { \babeltempb } })
6002     }&%
6003 \endgroup}
6004 \endgroup
6005 \def\bbl@activateposthyphen{&%

```



```

6006 \let\bbl@activateposthyphen\relax
6007 \directlua{
6008   Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
6009 }}
6010 \def\bbl@activateprehyphen{%
6011   \let\bbl@activateprehyphen\relax
6012   \directlua{
6013     Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
6014   }}

```

## 13.7 Layout

Unlike xetex, luatex requires only minimal changes for right-to-left layouts, particularly in monolingual documents (the engine itself reverses boxes – including column order or headings –, margins, etc.) with `bidi=basic`, without having to patch almost any macro where text direction is relevant.

`\@hangfrom` is useful in many contexts and it is redefined always with the `layout` option.

There are, however, a number of issues when the text direction is not the same as the box direction (as set by `\bodydir`), and when `\parbox` and `\hangindent` are involved. Fortunately, latest releases of luatex simplify a lot the solution with `\shapemode`.

With the issue #15 I realized commands are best patched, instead of redefined. With a few lines, a modification could be applied to several classes and packages. Now, `tabular` seems to work (at least in simple cases) with `array`, `tabularx`, `hhline`, `colortbl`, `longtable`, `booktabs`, etc. However, `dcolumn` still fails.

```

6015 \bbl@trace{Redefinitions for bidi layout}
6016 \ifx\@eqnnum\undefined\else
6017   \ifx\bbl@attr@dir\undefined\else
6018     \edef\@eqnnum{%
6019       \unexpanded{\ifcase\bbl@attr@dir\else\bbl@textdir\@ne\fi}%
6020       \unexpanded\expandafter{\@eqnnum}}
6021   \fi
6022 \fi
6023 \ifx\bbl@opt@layout\@nnil\endinput\fi % if no layout
6024 \ifnum\bbl@bidimode>\z@
6025   \def\bbl@nextfake#1{% non-local changes, use always inside a group!
6026     \bbl@exp{%
6027       \mathdir\the\bodydir
6028       #1%           Once entered in math, set boxes to restore values
6029       \<ifmmode>%
6030       \everyvbox{%
6031         \the\everyvbox
6032         \bodydir\the\bodydir
6033         \mathdir\the\mathdir
6034         \everyhbox{\the\everyhbox}%
6035         \everyvbox{\the\everyvbox}}%
6036       \everyhbox{%
6037         \the\everyhbox
6038         \bodydir\the\bodydir
6039         \mathdir\the\mathdir
6040         \everyhbox{\the\everyhbox}%
6041         \everyvbox{\the\everyvbox}}%
6042       \<fi>}}%
6043   \def\@hangfrom#1{%
6044     \setbox\@tempboxa\hbox{#1}%
6045     \hangindent\wd\@tempboxa
6046     \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
6047       \shapemode\@ne
6048     \fi

```

```

6049 \noindent\box\@tempboxa}
6050 \fi
6051 \IfBabelLayout{tabular}
6052 {\let\bbl@OL@tabular\@tabular
6053 \bbl@replace\@tabular{$}\bbl@nextfake$}%
6054 \let\bbl@NL@tabular\@tabular
6055 \AtBeginDocument{%
6056 \ifx\bbl@NL@tabular\@tabular\else
6057 \bbl@replace\@tabular{$}\bbl@nextfake$}%
6058 \let\bbl@NL@tabular\@tabular
6059 \fi}}
6060 {}
6061 \IfBabelLayout{lists}
6062 {\let\bbl@OL@list\list
6063 \bbl@sreplace\list{\parshape}\bbl@listparshape}%
6064 \let\bbl@NL@list\list
6065 \def\bbl@listparshape#1#2#3{%
6066 \parshape #1 #2 #3 %
6067 \ifnum\bbl@getluadir{page}=\bbl@getluadir{par}\else
6068 \shapemode\tw@
6069 \fi}}
6070 {}
6071 \IfBabelLayout{graphics}
6072 {\let\bbl@pictresetdir\relax
6073 \def\bbl@pictsetdir#1{%
6074 \ifcase\bbl@thetextdir
6075 \let\bbl@pictresetdir\relax
6076 \else
6077 \ifcase#1\bodydir TLT % Remember this sets the inner boxes
6078 \or\textdir TLT
6079 \else\bodydir TLT \textdir TLT
6080 \fi
6081 % \text\par\dir required in pgf:
6082 \def\bbl@pictresetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}%
6083 \fi}%
6084 \ifx\AddToHook\undefined\else
6085 \AddToHook{env/picture/begin}{\bbl@pictsetdir\tw@}%
6086 \directlua{
6087 Babel.get_picture_dir = true
6088 Babel.picture_has_bidi = 0
6089 function Babel.picture_dir (head)
6090 if not Babel.get_picture_dir then return head end
6091 for item in node.traverse(head) do
6092 if item.id == node.id'glyph' then
6093 local itemchar = item.char
6094 % TODO. Copy paste pattern from Babel.bidi (-r)
6095 local chardata = Babel.characters[itemchar]
6096 local dir = chardata and chardata.d or nil
6097 if not dir then
6098 for nn, et in ipairs(Babel.ranges) do
6099 if itemchar < et[1] then
6100 break
6101 elseif itemchar <= et[2] then
6102 dir = et[3]
6103 break
6104 end
6105 end
6106 end
6107 if dir and (dir == 'al' or dir == 'r') then

```

```

6108         Babel.picture_has_bidi = 1
6109     end
6110 end
6111 end
6112 return head
6113 end
6114 luatexbase.add_to_callback("hpack_filter", Babel.picture_dir,
6115     "Babel.picture_dir")
6116 }%
6117 \AtBeginDocument{%
6118     \long\def\put(#1,#2)#3{%
6119         \@killglue
6120         % Try:
6121         \ifx\bbl@pictresetdir\relax
6122             \def\bbl@tempc{0}%
6123         \else
6124             \directlua{
6125                 Babel.get_picture_dir = true
6126                 Babel.picture_has_bidi = 0
6127             }%
6128             \setbox\z@\hb@xt@\z@{%
6129                 \@defaultunitsset\@tempdimc{#1}\unitlength
6130                 \kern\@tempdimc
6131                 #3\hss}%
6132             \edef\bbl@tempc{\directlua{tex.print(Babel.picture_has_bidi)}}%
6133         \fi
6134         % Do:
6135         \@defaultunitsset\@tempdimc{#2}\unitlength
6136         \raise\@tempdimc\hb@xt@\z@{%
6137             \@defaultunitsset\@tempdimc{#1}\unitlength
6138             \kern\@tempdimc
6139             {\ifnum\bbl@tempc>\z@\bbl@pictresetdir\fi#3}\hss}%
6140         \ignorespaces}%
6141         \MakeRobust\put}%
6142 \fi
6143 \AtBeginDocument
6144 {
6145     \ifx\tikz@atbegin@node\undefined\else
6146         \ifx\AddToHook\undefined\else % TODO. Still tentative.
6147             \AddToHook{env/pgfpicture/begin}{\bbl@pictsetdir\@ne}%
6148             \bbl@add\pgfinterruptpicture{\bbl@pictresetdir}%
6149             \fi
6150             \let\bbl@OL\pgfpicture\pgfpicture
6151             \bbl@sreplace\pgfpicture{\pgfpicturetrue}%
6152             {\bbl@pictsetdir\z@\pgfpicturetrue}%
6153             \bbl@add\pgfsys@beginpicture{\bbl@pictsetdir\z@}%
6154             \bbl@add\tikz@atbegin@node{\bbl@pictresetdir}%
6155             \bbl@sreplace\tikz{\beginpgroup}%
6156             {\beginpgroup\bbl@pictsetdir\tw@}%
6157             \fi
6158             \ifx\AddToHook\undefined\else
6159                 \AddToHook{env/tcolorbox/begin}{\bbl@pictsetdir\@ne}%
6160             \fi
6161         }%

```

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.

```

6162 \IfBabelLayout{counters}%

```

```

6163 {\let\bbl@OL@@textsuperscript\textsuperscript
6164 \bbl@sreplace\textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
6165 \let\bbl@latinarabic=\@arabic
6166 \let\bbl@OL@@arabic\@arabic
6167 \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
6168 \@ifpackagewith{babel}{bidi=default}%
6169 {\let\bbl@asciroman=\@roman
6170 \let\bbl@OL@@roman\@roman
6171 \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
6172 \let\bbl@asciiRoman=\@Roman
6173 \let\bbl@OL@@roman\@Roman
6174 \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}%
6175 \let\bbl@OL@labelenumii\labelenumii
6176 \def\labelenumii{}\theenumii(}%
6177 \let\bbl@OL@p@enumiii\p@enumiii
6178 \def\p@enumiii{\p@enumii)\theenumii(}}{}}{}}
6179 <<Footnote changes>>
6180 \IfBabelLayout{footnotes}%
6181 {\let\bbl@OL@footnote\footnote
6182 \BabelFootnote\footnote\languagename{}}{}%
6183 \BabelFootnote\localfootnote\languagename{}}{}%
6184 \BabelFootnote\mainfootnote{}}{}}{}}
6185 {}

```

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

```

6186 \IfBabelLayout{extras}%
6187 {\let\bbl@OL@underline\underline
6188 \bbl@sreplace\underline{$\@@underline}{\bbl@nextfake$\@@underline}%
6189 \let\bbl@OL@LaTeX2e\LaTeX2e
6190 \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
6191 \if b\expandafter\@car\@series\@nil\boldmath\fi
6192 \babelsublr{%
6193 \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}}}
6194 {}
6195 </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):

Arrrrgh!! The UAX#9 algorithm is too deeply entrenched in the assumption of batch-style processing [...]. May the fleas of a thousand camels infest the armpits of those who design

supposedly general-purpose algorithms by looking at their own implementations, and fail to consider other possible implementations!

Well, it took me some time to guess what the batch rules in UAX#9 actually mean (in other word, *what* they do and *why*, and not only *how*), but I think (or I hope) I've managed to understand them. In some sense, there are two bidi modes, one for numbers, and the other for text. Furthermore, setting just the direction in R text is not enough, because there are actually *two* R modes (set explicitly in Unicode with RLM and ALM). In babel the dir is set by a higher protocol based on the language/script, which in turn sets the correct dir (<l>, <r> or <al>).

From UAX#9: "Where available, markup should be used instead of the explicit formatting characters". So, this simple version just ignores formatting characters. Actually, most of that annex is devoted to how to handle them.

BD14-BD16 are not implemented. Unicode (and the W3C) are making a great effort to deal with some special problematic cases in "streamed" plain text. I don't think this is the way to go – particular issues should be fixed by a high level interface taking into account the needs of the document. And here is where luatex excels, because everything related to bidi writing is under our control.

```
6196 (*basic-r)
6197 Babel = Babel or {}
6198
6199 Babel.bidi_enabled = true
6200
6201 require('babel-data-bidi.lua')
6202
6203 local characters = Babel.characters
6204 local ranges = Babel.ranges
6205
6206 local DIR = node.id("dir")
6207
6208 local function dir_mark(head, from, to, outer)
6209   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
6210   local d = node.new(DIR)
6211   d.dir = '+' .. dir
6212   node.insert_before(head, from, d)
6213   d = node.new(DIR)
6214   d.dir = '-' .. dir
6215   node.insert_after(head, to, d)
6216 end
6217
6218 function Babel.bidi(head, ispar)
6219   local first_n, last_n          -- first and last char with nums
6220   local last_es                 -- an auxiliary 'last' used with nums
6221   local first_d, last_d         -- first and last char in L/R block
6222   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):

```
6223   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
6224   local strong_lr = (strong == 'l') and 'l' or 'r'
6225   local outer = strong
6226
6227   local new_dir = false
6228   local first_dir = false
6229   local inmath = false
6230
6231   local last_lr
6232
6233   local type_n = ''
6234
```

```

6235 for item in node.traverse(head) do
6236
6237   -- three cases: glyph, dir, otherwise
6238   if item.id == node.id'glyph'
6239     or (item.id == 7 and item.subtype == 2) then
6240
6241     local itemchar
6242     if item.id == 7 and item.subtype == 2 then
6243       itemchar = item.replace.char
6244     else
6245       itemchar = item.char
6246     end
6247     local chardata = characters[itemchar]
6248     dir = chardata and chardata.d or nil
6249     if not dir then
6250       for nn, et in ipairs(ranges) do
6251         if itemchar < et[1] then
6252           break
6253         elseif itemchar <= et[2] then
6254           dir = et[3]
6255           break
6256         end
6257       end
6258     end
6259     dir = dir or 'l'
6260     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).

```

6261   if new_dir then
6262     attr_dir = 0
6263     for at in node.traverse(item.attr) do
6264       if at.number == luatexbase.registernumber'bbl@attr@dir' then
6265         attr_dir = at.value % 3
6266       end
6267     end
6268     if attr_dir == 1 then
6269       strong = 'r'
6270     elseif attr_dir == 2 then
6271       strong = 'al'
6272     else
6273       strong = 'l'
6274     end
6275     strong_lr = (strong == 'l') and 'l' or 'r'
6276     outer = strong_lr
6277     new_dir = false
6278   end
6279
6280   if dir == 'nsm' then dir = strong end -- W1

```

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

```

6281   dir_real = dir -- We need dir_real to set strong below
6282   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:

```

6283     if strong == 'al' then
6284         if dir == 'en' then dir = 'an' end          -- W2
6285         if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
6286         strong_lr = 'r'                             -- W3
6287     end

```

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

```

6288     elseif item.id == node.id'dir' and not inmath then
6289         new_dir = true
6290         dir = nil
6291     elseif item.id == node.id'math' then
6292         inmath = (item.subtype == 0)
6293     else
6294         dir = nil          -- Not a char
6295     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>.

```

6296     if dir == 'en' or dir == 'an' or dir == 'et' then
6297         if dir ~= 'et' then
6298             type_n = dir
6299         end
6300         first_n = first_n or item
6301         last_n = last_es or item
6302         last_es = nil
6303     elseif dir == 'es' and last_n then -- W3+W6
6304         last_es = item
6305     elseif dir == 'cs' then          -- it's right - do nothing
6306     elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
6307         if strong_lr == 'r' and type_n ~= '' then
6308             dir_mark(head, first_n, last_n, 'r')
6309         elseif strong_lr == 'l' and first_d and type_n == 'an' then
6310             dir_mark(head, first_n, last_n, 'r')
6311             dir_mark(head, first_d, last_d, outer)
6312             first_d, last_d = nil, nil
6313         elseif strong_lr == 'l' and type_n ~= '' then
6314             last_d = last_n
6315         end
6316         type_n = ''
6317         first_n, last_n = nil, nil
6318     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:

```

6319     if dir == 'l' or dir == 'r' then
6320         if dir ~= outer then
6321             first_d = first_d or item
6322             last_d = item
6323         elseif first_d and dir ~= strong_lr then
6324             dir_mark(head, first_d, last_d, outer)
6325             first_d, last_d = nil, nil
6326         end
6327     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>, resp’tly, 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.

```

6328   if dir and not last_lr and dir ~= 'l' and outer == 'r' then
6329       item.char = characters[item.char] and
6330           characters[item.char].m or item.char
6331   elseif (dir or new_dir) and last_lr ~= item then
6332       local mir = outer .. strong_lr .. (dir or outer)
6333       if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
6334           for ch in node.traverse(node.next(last_lr)) do
6335               if ch == item then break end
6336               if ch.id == node.id'glyph' and characters[ch.char] then
6337                   ch.char = characters[ch.char].m or ch.char
6338               end
6339           end
6340       end
6341   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).

```

6342   if dir == 'l' or dir == 'r' then
6343       last_lr = item
6344       strong = dir_real           -- Don't search back - best save now
6345       strong_lr = (strong == 'l') and 'l' or 'r'
6346   elseif new_dir then
6347       last_lr = nil
6348   end
6349 end

```

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

```

6350 if last_lr and outer == 'r' then
6351   for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
6352     if characters[ch.char] then
6353       ch.char = characters[ch.char].m or ch.char
6354     end
6355   end
6356 end
6357 if first_n then
6358   dir_mark(head, first_n, last_n, outer)
6359 end
6360 if first_d then
6361   dir_mark(head, first_d, last_d, outer)
6362 end

```

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

```

6363 return node.prev(head) or head
6364 end
6365 </basic-r>

```

And here the Lua code for bidi=basic:

```

6366 (*basic)
6367 Babel = Babel or {}
6368
6369 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
6370
6371 Babel.fontmap = Babel.fontmap or {}

```



```

6372 Babel.fontmap[0] = {}      -- l
6373 Babel.fontmap[1] = {}      -- r
6374 Babel.fontmap[2] = {}      -- al/an
6375
6376 Babel.bidi_enabled = true
6377 Babel.mirroring_enabled = true
6378
6379 require('babel-data-bidi.lua')
6380
6381 local characters = Babel.characters
6382 local ranges = Babel.ranges
6383
6384 local DIR = node.id('dir')
6385 local GLYPH = node.id('glyph')
6386
6387 local function insert_implicit(head, state, outer)
6388   local new_state = state
6389   if state.sim and state.eim and state.sim ~= state.eim then
6390     dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
6391     local d = node.new(DIR)
6392     d.dir = '+' .. dir
6393     node.insert_before(head, state.sim, d)
6394     local d = node.new(DIR)
6395     d.dir = '-' .. dir
6396     node.insert_after(head, state.eim, d)
6397   end
6398   new_state.sim, new_state.eim = nil, nil
6399   return head, new_state
6400 end
6401
6402 local function insert_numeric(head, state)
6403   local new
6404   local new_state = state
6405   if state.san and state.ean and state.san ~= state.ean then
6406     local d = node.new(DIR)
6407     d.dir = '+TLT'
6408     _, new = node.insert_before(head, state.san, d)
6409     if state.san == state.sim then state.sim = new end
6410     local d = node.new(DIR)
6411     d.dir = '-TLT'
6412     _, new = node.insert_after(head, state.ean, d)
6413     if state.ean == state.eim then state.eim = new end
6414   end
6415   new_state.san, new_state.ean = nil, nil
6416   return head, new_state
6417 end
6418
6419 -- TODO - \hbox with an explicit dir can lead to wrong results
6420 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
6421 -- was s made to improve the situation, but the problem is the 3-dir
6422 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
6423 -- well.
6424
6425 function Babel.bidi(head, ispar, hdir)
6426   local d -- d is used mainly for computations in a loop
6427   local prev_d = ''
6428   local new_d = false
6429
6430   local nodes = {}

```

```

6431 local outer_first = nil
6432 local inmath = false
6433
6434 local glue_d = nil
6435 local glue_i = nil
6436
6437 local has_en = false
6438 local first_et = nil
6439
6440 local ATDIR = luatexbase.registernumber'bbl@attr@dir'
6441
6442 local save_outer
6443 local temp = node.get_attribute(head, ATDIR)
6444 if temp then
6445     temp = temp % 3
6446     save_outer = (temp == 0 and 'l') or
6447                  (temp == 1 and 'r') or
6448                  (temp == 2 and 'al')
6449 elseif ispar then -- Or error? Shouldn't happen
6450     save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
6451 else -- Or error? Shouldn't happen
6452     save_outer = ('TRT' == hdir) and 'r' or 'l'
6453 end
6454 -- when the callback is called, we are just _after_ the box,
6455 -- and the textdir is that of the surrounding text
6456 -- if not ispar and hdir ~= tex.textdir then
6457 --     save_outer = ('TRT' == hdir) and 'r' or 'l'
6458 -- end
6459 local outer = save_outer
6460 local last = outer
6461 -- 'al' is only taken into account in the first, current loop
6462 if save_outer == 'al' then save_outer = 'r' end
6463
6464 local fontmap = Babel.fontmap
6465
6466 for item in node.traverse(head) do
6467
6468     -- In what follows, #node is the last (previous) node, because the
6469     -- current one is not added until we start processing the neutrals.
6470
6471     -- three cases: glyph, dir, otherwise
6472     if item.id == GLYPH
6473         or (item.id == 7 and item.subtype == 2) then
6474
6475         local d_font = nil
6476         local item_r
6477         if item.id == 7 and item.subtype == 2 then
6478             item_r = item.replace -- automatic discs have just 1 glyph
6479         else
6480             item_r = item
6481         end
6482         local chardata = characters[item_r.char]
6483         d = chardata and chardata.d or nil
6484         if not d or d == 'nsm' then
6485             for nn, et in ipairs(ranges) do
6486                 if item_r.char < et[1] then
6487                     break
6488                 elseif item_r.char <= et[2] then
6489                     if not d then d = et[3]

```

```

6490         elseif d == 'nsm' then d_font = et[3]
6491         end
6492         break
6493     end
6494 end
6495 end
6496 d = d or 'l'
6497
6498 -- A short 'pause' in bidi for mapfont
6499 d_font = d_font or d
6500 d_font = (d_font == 'l' and 0) or
6501           (d_font == 'nsm' and 0) or
6502           (d_font == 'r' and 1) or
6503           (d_font == 'al' and 2) or
6504           (d_font == 'an' and 2) or nil
6505 if d_font and fontmap and fontmap[d_font][item_r.font] then
6506     item_r.font = fontmap[d_font][item_r.font]
6507 end
6508
6509 if new_d then
6510     table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
6511     if inmath then
6512         attr_d = 0
6513     else
6514         attr_d = node.get_attribute(item, ATDIR)
6515         attr_d = attr_d % 3
6516     end
6517     if attr_d == 1 then
6518         outer_first = 'r'
6519         last = 'r'
6520     elseif attr_d == 2 then
6521         outer_first = 'r'
6522         last = 'al'
6523     else
6524         outer_first = 'l'
6525         last = 'l'
6526     end
6527     outer = last
6528     has_en = false
6529     first_et = nil
6530     new_d = false
6531 end
6532
6533 if glue_d then
6534     if (d == 'l' and 'l' or 'r') ~= glue_d then
6535         table.insert(nodes, {glue_i, 'on', nil})
6536     end
6537     glue_d = nil
6538     glue_i = nil
6539 end
6540
6541 elseif item.id == DIR then
6542     d = nil
6543     new_d = true
6544
6545 elseif item.id == node.id'glue' and item.subtype == 13 then
6546     glue_d = d
6547     glue_i = item
6548     d = nil

```

```

6549
6550 elseif item.id == node.id'math' then
6551     inmath = (item.subtype == 0)
6552
6553 else
6554     d = nil
6555 end
6556
6557 -- AL <= EN/ET/ES      -- W2 + W3 + W6
6558 if last == 'al' and d == 'en' then
6559     d = 'an'           -- W3
6560 elseif last == 'al' and (d == 'et' or d == 'es') then
6561     d = 'on'           -- W6
6562 end
6563
6564 -- EN + CS/ES + EN      -- W4
6565 if d == 'en' and #nodes >= 2 then
6566     if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
6567         and nodes[#nodes-1][2] == 'en' then
6568         nodes[#nodes][2] = 'en'
6569     end
6570 end
6571
6572 -- AN + CS + AN         -- W4 too, because uax9 mixes both cases
6573 if d == 'an' and #nodes >= 2 then
6574     if (nodes[#nodes][2] == 'cs')
6575         and nodes[#nodes-1][2] == 'an' then
6576         nodes[#nodes][2] = 'an'
6577     end
6578 end
6579
6580 -- ET/EN                -- W5 + W7->l / W6->on
6581 if d == 'et' then
6582     first_et = first_et or (#nodes + 1)
6583 elseif d == 'en' then
6584     has_en = true
6585     first_et = first_et or (#nodes + 1)
6586 elseif first_et then    -- d may be nil here !
6587     if has_en then
6588         if last == 'l' then
6589             temp = 'l'    -- W7
6590         else
6591             temp = 'en'   -- W5
6592         end
6593     else
6594         temp = 'on'       -- W6
6595     end
6596     for e = first_et, #nodes do
6597         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
6598     end
6599     first_et = nil
6600     has_en = false
6601 end
6602
6603 -- Force mathdir in math if ON (currently works as expected only
6604 -- with 'l')
6605 if inmath and d == 'on' then
6606     d = ('TRT' == tex.mathdir) and 'r' or 'l'
6607 end

```

```

6608
6609     if d then
6610         if d == 'al' then
6611             d = 'r'
6612             last = 'al'
6613         elseif d == 'l' or d == 'r' then
6614             last = d
6615         end
6616         prev_d = d
6617         table.insert(nodes, {item, d, outer_first})
6618     end
6619
6620     outer_first = nil
6621
6622 end
6623
6624 -- TODO -- repeated here in case EN/ET is the last node. Find a
6625 -- better way of doing things:
6626 if first_et then      -- dir may be nil here !
6627     if has_en then
6628         if last == 'l' then
6629             temp = 'l'    -- W7
6630         else
6631             temp = 'en'   -- W5
6632         end
6633     else
6634         temp = 'on'       -- W6
6635     end
6636     for e = first_et, #nodes do
6637         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
6638     end
6639 end
6640
6641 -- dummy node, to close things
6642 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
6643
6644 ----- NEUTRAL -----
6645
6646 outer = save_outer
6647 last = outer
6648
6649 local first_on = nil
6650
6651 for q = 1, #nodes do
6652     local item
6653
6654     local outer_first = nodes[q][3]
6655     outer = outer_first or outer
6656     last = outer_first or last
6657
6658     local d = nodes[q][2]
6659     if d == 'an' or d == 'en' then d = 'r' end
6660     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
6661
6662     if d == 'on' then
6663         first_on = first_on or q
6664     elseif first_on then
6665         if last == d then
6666             temp = d

```

```

6667     else
6668         temp = outer
6669     end
6670     for r = first_on, q - 1 do
6671         nodes[r][2] = temp
6672         item = nodes[r][1]    -- MIRRORING
6673         if Babel.mirroring_enabled and item.id == GLYPH
6674             and temp == 'r' and characters[item.char] then
6675             local font_mode = font.fonts[item.font].properties.mode
6676             if font_mode ~= 'harf' and font_mode ~= 'plug' then
6677                 item.char = characters[item.char].m or item.char
6678             end
6679         end
6680     end
6681     first_on = nil
6682 end
6683
6684     if d == 'r' or d == 'l' then last = d end
6685 end
6686
6687 ----- IMPLICIT, REORDER -----
6688
6689 outer = save_outer
6690 last = outer
6691
6692 local state = {}
6693 state.has_r = false
6694
6695 for q = 1, #nodes do
6696
6697     local item = nodes[q][1]
6698
6699     outer = nodes[q][3] or outer
6700
6701     local d = nodes[q][2]
6702
6703     if d == 'nsm' then d = last end          -- W1
6704     if d == 'en' then d = 'an' end
6705     local isdir = (d == 'r' or d == 'l')
6706
6707     if outer == 'l' and d == 'an' then
6708         state.san = state.san or item
6709         state.ean = item
6710     elseif state.san then
6711         head, state = insert_numeric(head, state)
6712     end
6713
6714     if outer == 'l' then
6715         if d == 'an' or d == 'r' then      -- im -> implicit
6716             if d == 'r' then state.has_r = true end
6717             state.sim = state.sim or item
6718             state.eim = item
6719         elseif d == 'l' and state.sim and state.has_r then
6720             head, state = insert_implicit(head, state, outer)
6721         elseif d == 'l' then
6722             state.sim, state.eim, state.has_r = nil, nil, false
6723         end
6724     else
6725         if d == 'an' or d == 'l' then

```

```

6726         if nodes[q][3] then -- nil except after an explicit dir
6727             state.sim = item -- so we move sim 'inside' the group
6728         else
6729             state.sim = state.sim or item
6730         end
6731         state.eim = item
6732     elseif d == 'r' and state.sim then
6733         head, state = insert_implicit(head, state, outer)
6734     elseif d == 'r' then
6735         state.sim, state.eim = nil, nil
6736     end
6737 end
6738
6739 if isdir then
6740     last = d -- Don't search back - best save now
6741 elseif d == 'on' and state.san then
6742     state.san = state.san or item
6743     state.ean = item
6744 end
6745
6746 end
6747
6748 return node.prev(head) or head
6749 end
6750 </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.

```

6751 <{*nil}
6752 \ProvidesLanguage{nil}[<<date>>] <<version>> Nil language]
6753 \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.

```

6754 \ifx\l@nil\undefined
6755     \newlanguage\l@nil
6756     \@namedef{bbl@hyphendata@the\l@nil}{{}}{}% Remove warning
6757     \let\bbl@elt\relax
6758     \edef\bbl@languages{% Add it to the list of languages

```

```
6759 \bbl@languages\bbl@elt{nil}{\the\l@nil}{\}}
6760 \fi
```

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

```
6761 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}
```

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

```
\captionnil
\datenil
6762 \let\captionnil\@empty
6763 \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.

```
6764 \ldf@finish{nil}
6765 </nil>
```

## 16 Support for Plain T<sub>E</sub>X (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 T<sub>E</sub>X-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 `locallyhyphen.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`.

```
6766 <{*bplain | blplain}
6767 \catcode`\{=1 % left brace is begin-group character
6768 \catcode`\}=2 % right brace is end-group character
6769 \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).

```
6770 \openin 0 hyphen.cfg
6771 \ifeof0
6772 \else
6773 \let\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.

```
6774 \def\input #1 {%
6775 \let\input\input
6776 \a hyphen.cfg
6777 \let\input\undefined
6778 }
6779 \fi
6780 </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`.

```
6781 \bplain\la plain.tex
6782 \bplain\la 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.

```
6783 \bplain\def\fmtname{babel-plain}
6784 \bplain\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 $\text{\LaTeX}$ features

The following code duplicates or emulates parts of  $\text{\LaTeX} 2_{\epsilon}$  that are needed for `babel`.

```
6785 \langle *Emulate LaTeX \rangle \equiv
6786 % == Code for plain ==
6787 \def\@empty{}
6788 \def\loadlocalcfg#1{%
6789   \openin0#1.cfg
6790   \ifeof0
6791     \closein0
6792   \else
6793     \closein0
6794     {\immediate\write16{*****}%
6795       \immediate\write16{* Local config file #1.cfg used}%
6796       \immediate\write16{*}%
6797     }
6798     \input #1.cfg\relax
6799   \fi
6800 \endofldf}
```

## 16.3 General tools

A number of  $\text{\LaTeX}$  macro's that are needed later on.

```
6801 \long\def\@firstofone#1{#1}
6802 \long\def\@firstoftwo#1#2{#1}
6803 \long\def\@secondoftwo#1#2{#2}
6804 \def\@nnil{\@nil}
6805 \def\@gobbletwo#1#2{}
6806 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
6807 \def\@star@or@long#1{%
6808   \@ifstar
6809   {\let\l@ngrel@x\relax#1}%
6810   {\let\l@ngrel@x\long#1}}
6811 \let\l@ngrel@x\relax
6812 \def\@car#1#2\@nil{#1}
6813 \def\@cdr#1#2\@nil{#2}
6814 \let\@typeset@protect\relax
6815 \let\protected@edef\edef
6816 \long\def\@gobble#1{}
6817 \edef\@backslashchar{\expandafter\@gobble\string\}
6818 \def\strip@prefix#1>{}
6819 \def\g@addto@macro#1#2{%
6820   \toks@\expandafter{#1#2}%
6821   \xdef#1{\the\toks@}}
6822 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
```

```

6823 \def\@nameuse#1{\csname #1\endcsname}
6824 \def\@ifundefined#1{%
6825   \expandafter\ifx\csname#1\endcsname\relax
6826     \expandafter\@firstoftwo
6827   \else
6828     \expandafter\@secondoftwo
6829   \fi}
6830 \def\@expandtwoargs#1#2#3{%
6831   \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
6832 \def\zap@space#1 #2{%
6833   #1%
6834   \ifx#2\@empty\else\expandafter\zap@space\fi
6835   #2}
6836 \let\bbl@trace\@gobble

```

$\LaTeX_{\epsilon}$  has the command `\@onlypreamble` which adds commands to a list of commands that are no longer needed after `\begin{document}`.

```

6837 \ifx\@preamblecmds\@undefined
6838   \def\@preamblecmds{}
6839 \fi
6840 \def\@onlypreamble#1{%
6841   \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
6842     \@preamblecmds\do#1}}
6843 \@onlypreamble\@onlypreamble

```

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

```

6844 \def\begindocument{%
6845   \@begindocumenthook
6846   \global\let\@begindocumenthook\@undefined
6847   \def\do##1{\global\let##1\@undefined}%
6848   \@preamblecmds
6849   \global\let\do\noexpand}
6850 \ifx\@begindocumenthook\@undefined
6851   \def\@begindocumenthook{}
6852 \fi
6853 \@onlypreamble\@begindocumenthook
6854 \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`.

```

6855 \def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
6856 \@onlypreamble\AtEndOfPackage
6857 \def\@endofldf{}
6858 \@onlypreamble\@endofldf
6859 \let\bbl@afterlang\@empty
6860 \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.

```

6861 \catcode`\&=\z@
6862 \ifx&\if@files\@undefined
6863   \expandafter\let\csname if@files\endcsname\expandafter\endcsname
6864   \csname iffalse\endcsname
6865 \fi
6866 \catcode`\&=4

```

Mimick  $\LaTeX$ 's commands to define control sequences.

```

6867 \def\newcommand{\@star@or@long\newcommand}

```

```

6868 \def\new@command#1{%
6869   \@testopt{\@newcommand#1}0}
6870 \def\@newcommand#1[#2]{%
6871   \@ifnextchar [{\@xargdef#1[#2]}%
6872                 {\@argdef#1[#2]}}
6873 \long\def\@argdef#1[#2]#3{%
6874   \@yargdef#1\@ne{#2}{#3}}
6875 \long\def\@xargdef#1[#2][#3]#4{%
6876   \expandafter\def\expandafter#1\expandafter{%
6877     \expandafter\@protected@testopt\expandafter #1%
6878     \csname\string#1\expandafter\endcsname{#3}}}%
6879   \expandafter\@yargdef \csname\string#1\endcsname
6880   \tw@{#2}{#4}}
6881 \long\def\@yargdef#1#2#3{%
6882   \@tempcnta#3\relax
6883   \advance \@tempcnta \@ne
6884   \let\@hash@\relax
6885   \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
6886   \@tempcntb #2%
6887   \@whilenum\@tempcntb <\@tempcnta
6888   \do{%
6889     \edef\reserved@a{\reserved@a\@hash@\the\@tempcntb}%
6890     \advance\@tempcntb \@ne}%
6891   \let\@hash@###
6892   \l@ngrel@x\expandafter\def\expandafter#1\reserved@a}
6893 \def\providecommand{\@star@or@long\provide@command}
6894 \def\provide@command#1{%
6895   \begingroup
6896     \escapechar\m@ne\xdef\@gtempa{{\string#1}}%
6897   \endgroup
6898   \expandafter\@ifundefined\@gtempa
6899     {\def\reserved@a{\new@command#1}}%
6900     {\let\reserved@a\relax
6901      \def\reserved@a{\new@command\reserved@a}}%
6902   \reserved@a}%

6903 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
6904 \def\declare@robustcommand#1{%
6905   \edef\reserved@a{\string#1}%
6906   \def\reserved@b{#1}%
6907   \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
6908   \edef#1{%
6909     \ifx\reserved@a\reserved@b
6910       \noexpand\x@protect
6911       \noexpand#1%
6912     \fi
6913     \noexpand\protect
6914     \expandafter\noexpand\csname
6915       \expandafter\@gobble\string#1 \endcsname
6916   }%
6917   \expandafter\new@command\csname
6918     \expandafter\@gobble\string#1 \endcsname
6919 }
6920 \def\x@protect#1{%
6921   \ifx\protect\@typeset@protect\else
6922     \@x@protect#1%
6923   \fi
6924 }
6925 \catcode`\&=\z@ % Trick to hide conditionals

```

```
6926 \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`.

```
6927 \def\bbl@tempa{\csname newif\endcsname&ifin@}
6928 \catcode`\&=4
6929 \ifx\in@\@undefined
6930 \def\in@#1#2{%
6931 \def\in@@##1##2##3\in@{%
6932 \ifx\in@@##2\in@false\else\in@true\fi}%
6933 \in@@#2#1\in@\in@@}
6934 \else
6935 \let\bbl@tempa\@empty
6936 \fi
6937 \bbl@tempa
```

$\text{\TeX}$  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  $\text{\TeX}$  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).

```
6938 \def\ifpackagewith#1#2#3#4{#3}
```

The  $\text{\TeX}$  macro `\ifl@aded` checks whether a file was loaded. This functionality is not needed for plain  $\text{\TeX}$  but we need the macro to be defined as a no-op.

```
6939 \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  $\text{\TeX}$  2 $\epsilon$  versions; just enough to make things work in plain  $\text{\TeX}$  environments.

```
6940 \ifx\@tempcnta\@undefined
6941 \csname newcount\endcsname\@tempcnta\relax
6942 \fi
6943 \ifx\@tempcntb\@undefined
6944 \csname newcount\endcsname\@tempcntb\relax
6945 \fi
```

To prevent wasting two counters in  $\text{\TeX}$  2.09 (because counters with the same name are allocated later by it) we reset the counter that holds the next free counter (`\count10`).

```
6946 \ifx\bye\@undefined
6947 \advance\count10 by -2\relax
6948 \fi
6949 \ifx\@ifnextchar\@undefined
6950 \def\@ifnextchar#1#2#3{%
6951 \let\reserved@d=#1%
6952 \def\reserved@a{#2}\def\reserved@b{#3}%
6953 \futurelet\@let@token\@ifnch}
6954 \def\@ifnch{%
6955 \ifx\@let@token\@sptoken
6956 \let\reserved@c\@xifnch
6957 \else
6958 \ifx\@let@token\reserved@d
6959 \let\reserved@c\reserved@a
6960 \else
6961 \let\reserved@c\reserved@b
6962 \fi
6963 \fi
6964 \reserved@c}
6965 \def\:{\let\@sptoken= } \: % this makes \@sptoken a space token
```

```

6966 \def\:{\@xifnch} \expandafter\def\:{\futurelet\@let@token\@ifnch}
6967 \fi
6968 \def\@testopt#1#2{%
6969 \@ifnextchar[{\#1}{\#1[\#2]}}
6970 \def\@protected@testopt#1{%
6971 \ifx\protect\@typeset@protect
6972 \expandafter\@testopt
6973 \else
6974 \@x@protect#1%
6975 \fi}
6976 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
6977 #2\relax}\fi}
6978 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
6979 \else\expandafter\@gobble\fi{#1}}

```

## 16.4 Encoding related macros

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

```

6980 \def\DeclareTextCommand{%
6981 \@dec@text@cmd\providecommand
6982 }
6983 \def\ProvideTextCommand{%
6984 \@dec@text@cmd\providecommand
6985 }
6986 \def\DeclareTextSymbol#1#2#3{%
6987 \@dec@text@cmd\chardef#1{#2}#3\relax
6988 }
6989 \def\@dec@text@cmd#1#2#3{%
6990 \expandafter\def\expandafter#2%
6991 \expandafter{%
6992 \csname#3-cmd\expandafter\endcsname
6993 \expandafter#2%
6994 \csname#3\string#2\endcsname
6995 }%
6996 % \let\@ifdefinable\@rc@ifdefinable
6997 \expandafter#1\csname#3\string#2\endcsname
6998 }
6999 \def\@current@cmd#1{%
7000 \ifx\protect\@typeset@protect\else
7001 \noexpand#1\expandafter\@gobble
7002 \fi
7003 }
7004 \def\@changed@cmd#1#2{%
7005 \ifx\protect\@typeset@protect
7006 \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
7007 \expandafter\ifx\csname ?\string#1\endcsname\relax
7008 \expandafter\def\csname ?\string#1\endcsname{%
7009 \@changed@x@err{#1}%
7010 }%
7011 \fi
7012 \global\expandafter\let
7013 \csname\cf@encoding\string#1\expandafter\endcsname
7014 \csname ?\string#1\endcsname
7015 \fi
7016 \csname\cf@encoding\string#1%
7017 \expandafter\endcsname
7018 \else
7019 \noexpand#1%

```

```

7020 \fi
7021 }
7022 \def\@changed@x@err#1{%
7023 \errhelp{Your command will be ignored, type <return> to proceed}%
7024 \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
7025 \def\DeclareTextCommandDefault#1{%
7026 \DeclareTextCommand#1?%
7027 }
7028 \def\ProvideTextCommandDefault#1{%
7029 \ProvideTextCommand#1?%
7030 }
7031 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
7032 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
7033 \def\DeclareTextAccent#1#2#3{%
7034 \DeclareTextCommand#1{#2}[1]{\accent#3 #1}
7035 }
7036 \def\DeclareTextCompositeCommand#1#2#3#4{%
7037 \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
7038 \edef\reserved@b{\string##1}%
7039 \edef\reserved@c{%
7040 \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
7041 \ifx\reserved@b\reserved@c
7042 \expandafter\expandafter\expandafter\ifx
7043 \expandafter\@car\reserved@a\relax\relax\@nil
7044 \@text@composite
7045 \else
7046 \edef\reserved@b##1{%
7047 \def\expandafter\noexpand
7048 \csname#2\string#1\endcsname####1{%
7049 \noexpand\@text@composite
7050 \expandafter\noexpand\csname#2\string#1\endcsname
7051 ####1\noexpand\@empty\noexpand\@text@composite
7052 {##1}%
7053 }%
7054 }%
7055 \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
7056 \fi
7057 \expandafter\def\csname\expandafter\string\csname
7058 #2\endcsname\string#1-\string#3\endcsname{#4}
7059 \else
7060 \errhelp{Your command will be ignored, type <return> to proceed}%
7061 \errmessage{\string\DeclareTextCompositeCommand\space used on
7062 inappropriate command \protect#1}
7063 \fi
7064 }
7065 \def\@text@composite#1#2#3\@text@composite{%
7066 \expandafter\@text@composite@x
7067 \csname\string#1-\string#2\endcsname
7068 }
7069 \def\@text@composite@x#1#2{%
7070 \ifx#1\relax
7071 #2%
7072 \else
7073 #1%
7074 \fi
7075 }
7076 %
7077 \def\@strip@args#1:#2-#3\@strip@args{#2}
7078 \def\DeclareTextComposite#1#2#3#4{%

```

```

7079 \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
7080 \bgroup
7081 \lccode` \@=#4%
7082 \lowercase{%
7083 \egroup
7084 \reserved@a @%
7085 }%
7086 }
7087 %
7088 \def\UseTextSymbol#1#2{#2}
7089 \def\UseTextAccent#1#2#3{}
7090 \def\@use@text@encoding#1{}
7091 \def\DeclareTextSymbolDefault#1#2{%
7092 \DeclareTextCommandDefault#1{\UseTextSymbol{#2}{#1}}%
7093 }
7094 \def\DeclareTextAccentDefault#1#2{%
7095 \DeclareTextCommandDefault#1{\UseTextAccent{#2}{#1}}%
7096 }
7097 \def\cf@encoding{OT1}

```

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

```

7098 \DeclareTextAccent{"}{OT1}{127}
7099 \DeclareTextAccent{'}{OT1}{19}
7100 \DeclareTextAccent{^}{OT1}{94}
7101 \DeclareTextAccent{`}{OT1}{18}
7102 \DeclareTextAccent{~}{OT1}{126}

```

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

```

7103 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
7104 \DeclareTextSymbol{\textquotedblright}{OT1}{`"}
7105 \DeclareTextSymbol{\textquoteleft}{OT1}{`'}
7106 \DeclareTextSymbol{\textquoteright}{OT1}{`'}
7107 \DeclareTextSymbol{\i}{OT1}{16}
7108 \DeclareTextSymbol{\ss}{OT1}{25}

```

For a couple of languages we need the  $\text{\LaTeX}$ -control sequence `\scriptsize` to be available. Because `plain TEX` doesn't have such a sophisticated font mechanism as  $\text{\LaTeX}$  has, we just `\let` it to `\sevenrm`.

```

7109 \ifx\scriptsize\@undefined
7110 \let\scriptsize\sevenrm
7111 \fi
7112 % End of code for plain
7113 <</Emulate LaTeX>>

```

A proxy file:

```

7114 <{*plain}>
7115 \input babel.def
7116 </plain>

```

## 17 Acknowledgements

I would like to thank all who volunteered as  $\beta$ -testers for their time. Michel Goossens supplied contributions for most of the other languages. Nico Poppelier helped polish the text of the documentation and supplied parts of the macros for the Dutch language. Paul Wackers and Werenfried Spit helped find and repair bugs.

During the further development of the `babel` system I received much help from Bernd Raichle, for which I am grateful.

## References

- [1] Huda Smitshuijzen Abifares, *Arabic Typography*, Saqi, 2001.
- [2] Johannes Braams, Victor Eijkhout and Nico Poppelier, *The development of national  $\text{\LaTeX}$  styles*, *TUGboat* 10 (1989) #3, p. 401–406.
- [3] Yannis Haralambous, *Fonts & Encodings*, O'Reilly, 2007.
- [4] Donald E. Knuth, *The  $\text{\TeX}$ book*, Addison-Wesley, 1986.
- [5] Jukka K. Korpela, *Unicode Explained*, O'Reilly, 2006.
- [6] Leslie Lamport,  *$\text{\LaTeX}$ , A document preparation System*, Addison-Wesley, 1986.
- [7] Leslie Lamport, in:  $\text{\TeX}$ hax Digest, Volume 89, #13, 17 February 1989.
- [8] Ken Lunde, *CJKV Information Processing*, O'Reilly, 2nd ed., 2009.
- [9] Hubert Partl, *German  $\text{\TeX}$* , *TUGboat* 9 (1988) #1, p. 70–72.
- [10] Joachim Schrod, *International  $\text{\LaTeX}$  is ready to use*, *TUGboat* 11 (1990) #1, p. 87–90.
- [11] Apostolos Syropoulos, Antonis Tsolomitis and Nick Sofroniu, *Digital typography using  $\text{\LaTeX}$* , Springer, 2002, p. 301–373.
- [12] K.F. Treebus. *Tekstwijzer, een gids voor het grafisch verwerken van tekst*, SDU Uitgeverij ('s-Gravenhage, 1988).