

# Babel

Version 3.63.2485  
2021/09/04

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 . . . . .	8
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 . . . . .	29
1.17	Digits and counters . . . . .	33
1.18	Dates . . . . .	34
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 . . . . .	47
1.28	Unicode character properties in luatex . . . . .	49
1.29	Tweaking some features . . . . .	49
1.30	Tips, workarounds, known issues and notes . . . . .	49
1.31	Current and future work . . . . .	50
1.32	Tentative and experimental code . . . . .	51
<b>2</b>	<b>Loading languages with language.dat</b>	<b>51</b>
2.1	Format . . . . .	51
<b>3</b>	<b>The interface between the core of babel and the language definition files</b>	<b>52</b>
3.1	Guidelines for contributed languages . . . . .	53
3.2	Basic macros . . . . .	54
3.3	Skeleton . . . . .	55
3.4	Support for active characters . . . . .	56
3.5	Support for saving macro definitions . . . . .	57
3.6	Support for extending macros . . . . .	57
3.7	Macros common to a number of languages . . . . .	57
3.8	Encoding-dependent strings . . . . .	57
<b>4</b>	<b>Changes</b>	<b>61</b>
4.1	Changes in babel version 3.9 . . . . .	61

<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>62</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	<code>base</code> . . . . .	69
7.4	<code>key=value</code> options and other general option . . . . .	69
7.5	Conditional loading of shorthands . . . . .	71
7.6	Interlude for Plain . . . . .	73
<b>8</b>	<b>Multiple languages</b>	<b>73</b>
8.1	Selecting the language . . . . .	76
8.2	Errors . . . . .	84
8.3	Hooks . . . . .	87
8.4	Setting up language files . . . . .	89
8.5	Shorthands . . . . .	91
8.6	Language attributes . . . . .	100
8.7	Support for saving macro definitions . . . . .	102
8.8	Short tags . . . . .	103
8.9	Hyphens . . . . .	104
8.10	Multiencoding strings . . . . .	105
8.11	Macros common to a number of languages . . . . .	112
8.12	Making glyphs available . . . . .	112
8.12.1	Quotation marks . . . . .	112
8.12.2	Letters . . . . .	114
8.12.3	Shorthands for quotation marks . . . . .	115
8.12.4	Umlauts and tremas . . . . .	115
8.13	Layout . . . . .	117
8.14	Load engine specific macros . . . . .	117
8.15	Creating and modifying languages . . . . .	117
<b>9</b>	<b>Adjusting the Babel behavior</b>	<b>139</b>
9.1	Cross referencing macros . . . . .	141
9.2	Marks . . . . .	143
9.3	Preventing clashes with other packages . . . . .	144
9.3.1	<code>ifthen</code> . . . . .	144
9.3.2	<code>varioref</code> . . . . .	145
9.3.3	<code>hhline</code> . . . . .	145
9.4	Encoding and fonts . . . . .	146
9.5	Basic bidi support . . . . .	148
9.6	Local Language Configuration . . . . .	151
9.7	Language options . . . . .	152
<b>10</b>	<b>The kernel of Babel (<code>babel.def</code>, <code>common</code>)</b>	<b>155</b>
<b>11</b>	<b>Loading hyphenation patterns</b>	<b>155</b>
<b>12</b>	<b>Font handling with <code>fontspec</code></b>	<b>160</b>

<b>13</b>	<b>Hooks for XeTeX and LuaTeX</b>	<b>164</b>
13.1	XeTeX . . . . .	164
13.2	Layout . . . . .	166
13.3	LuaTeX . . . . .	167
13.4	Southeast Asian scripts . . . . .	173
13.5	CJK line breaking . . . . .	175
13.6	Arabic justification . . . . .	177
13.7	Common stuff . . . . .	181
13.8	Automatic fonts and ids switching . . . . .	181
13.9	Bidi . . . . .	186
13.10	Layout . . . . .	188
13.11	Lua: transforms . . . . .	191
13.12	Lua: Auto bidi with basic and basic-r . . . . .	200
<b>14</b>	<b>Data for CJK</b>	<b>211</b>
<b>15</b>	<b>The ‘nil’ language</b>	<b>211</b>
<b>16</b>	<b>Support for Plain T<sub>E</sub>X (plain.def)</b>	<b>212</b>
16.1	Not renaming hyphen.tex . . . . .	212
16.2	Emulating some L <sup>A</sup> T <sub>E</sub> X features . . . . .	212
16.3	General tools . . . . .	213
16.4	Encoding related macros . . . . .	217
<b>17</b>	<b>Acknowledgements</b>	<b>220</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 . . . . .	8
Unknown language ‘LANG’ . . . . .	9
Argument of \language@active@arg” has an extra } . . . . .	12
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 `e-Plain` and `pdf-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 site](#). 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  $\TeX$  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  $\text{\LaTeX}$ , the preamble of the document:

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

would tell  $\text{\LaTeX}$  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 three-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.

### 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>

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

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

```
! 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 e-Plain and pdf-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 those formats. Please, refer to [Using babel with Plain](#) for further details.

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

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

```
\selectlanguage{german}
```

This command can be used as environment, too.

**NOTE** For “historical reasons”, a macro name is converted to a language name without the leading `\`; in other words, `\selectlanguage{\german}` is equivalent to `\selectlanguage{german}`. Using a macro instead of a “real” name is deprecated. **New 3.43** However, if the macro name does not match any language, it will get expanded as expected.

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

**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.

**WARNING** `\selectlanguage` should not be used inside some boxed environments (like floats or minipage) to switch the language if you need the information written to the aux be correctly synchronized. This rarely happens, but if it were the case, you must use `other language` instead.

**`\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), and since it is meant for phrases only the text direction (and not the paragraph one) is set.

**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 `other language` 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  $\TeX$  and conflicts with existing macros may arise (`\textlatin`, `\textbar`, `\textit`, `\textcolor` and many others). The same applies to environments, because `arabic` conflicts with `\arabic`. Furthermore, and because of this overloading, detecting the language of a chunk of text by external tools can become unfeasible. 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>4</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 discretionary 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`).

**TROUBLESHOOTING** A typical error when using shorthands is the following:

---

<sup>4</sup>With it, encoded strings may not work as expected.

! 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`  $*\{\langle shorthands-list \rangle\}$

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.

**WARNING** It is worth emphasizing these macros are meant for temporary changes. Whenever possible and if there are not conflicts with other packages, shorthands must be always enabled (or disabled).

`\usesshorthands`  $*\{\langle char \rangle\}$

The command `\usesshorthands` 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 `\usesshorthands*{\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 `\usesshorthands`. 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>5</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>5</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>6</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>7</sup>

`\ifbabelshorthand`  $\{\langle character \rangle\}\{\langle true \rangle\}\{\langle false \rangle\}$

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

`\aliasshorthand`  $\{\langle original \rangle\}\{\langle alias \rangle\}$

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, `\aliasshorthands` is ignored).

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

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

**WARNING** Shorthands remember somehow the original character, and the fallback value is that of the latter. So, in this example, if no shorthand 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.

<sup>6</sup>Thanks to Enrico Gregorio

<sup>7</sup>This declaration serves to nothing, but it is preserved for backward compatibility.



## 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  $\LaTeX$  before they are passed to the package and therefore they will not be recognized); however, t is provided for the common case of ~ (as well as c for not so common case of the comma). With shorthands=off no language shorthands are defined, As some languages use this mechanism for tools not available otherwise, a macro \babelshorthand is defined, which allows using them; see above.

**safe=** none | ref | bib  
Some  $\LaTeX$  macros are redefined so that using shorthands is safe. With safe=bib only \nocite, \bibcite and \bibitem are redefined. With safe=ref only \newlabel, \ref and \pageref are redefined (as well as a few macros from varioref and ifthen). With safe=none no macro is redefined. This option is strongly recommended, because a good deal of incompatibilities and errors are related to these redefinitions. As of **New 3.34**, in  $\epsilon\TeX$  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).

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

- headfoot=** `<language>`
- By default, headlines and footlines are not touched (only marks), and if they contain language-dependent macros (which is not usual) there may be unexpected results. With this option you may set the language in heads and foots.
- noconfigs** Global and language default config files are not loaded, so you can make sure your document is not spoilt by an unexpected `.cfg` file. However, if the key config is set, this file is loaded.
- showlanguages** Prints to the log the list of languages loaded when the format was created: number (remember dialects can share it), name, hyphenation file and exceptions file.
- nocase** New 3.9l Language settings for uppercase and lowercase mapping (as set by `\SetCase`) are ignored. Use only if there are incompatibilities with other packages.
- silent** New 3.9l No warnings and no *infos* are written to the log file.<sup>8</sup>
- strings=** `generic` | `unicode` | `encoded` | `<label>` | `<font encoding>`
- Selects the encoding of strings in languages supporting this feature. Predefined labels are `generic` (for traditional  $\TeX$ , LICR and ASCII strings), `unicode` (for engines like xetex and luatex) and `encoded` (for special cases requiring mixed encodings). Other allowed values are font encoding codes (T1, T2A, LGR, L7X...), but only in languages supporting them. Be aware with encoded captions are protected, but they work in `\MakeUppercase` and the like (this feature misuses some internal  $\LaTeX$  tools, so use it only as a last resort).
- hyphenmap=** `off` | `first` | `select` | `other` | `other*`
- New 3.9g Sets the behavior of case mapping for hyphenation, provided the language defines it.<sup>9</sup> It can take the following values:
- off** deactivates this feature and no case mapping is applied;
- first** sets it at the first switching commands in the current or parent scope (typically, when the aux file is first read and at `\begin{document}`}, but also the first `\selectlanguage` in the preamble), and it's the default if a single language option has been stated.<sup>10</sup>
- select** sets it only at `\selectlanguage`;
- other** also sets it at `otherlanguage`;
- other\*** also sets it at `otherlanguage*` as well as in heads and foots (if the option `headfoot` is used) and in auxiliary files (ie, at `\select@language`), and it's the default if several language options have been stated. The option `first` can be regarded as an optimized version of `other*` for monolingual documents.<sup>11</sup>
- bidi=** `default` | `basic` | `basic-r` | `bidi-l` | `bidi-r`
- New 3.14 Selects the bidi algorithm to be used in luatex and xetex. See sec. 1.24.
- layout=** New 3.16 Selects which layout elements are adapted in bidi documents. See sec. 1.24.

<sup>8</sup>You can use alternatively the package `silence`.

<sup>9</sup>Turned off in plain.

<sup>10</sup>Duplicated options count as several ones.

<sup>11</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] think it isn't really useful, but who knows.

## 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}\dots}
```

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 Locale Data Repository, which was used as source for most of the data provided by these files, too (the main exception being the `\dots` name strings).

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

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

LUATEX/XETEX

```
\documentclass{book}

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

```

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

\begin{document}

\tableofcontents

\chapter{სამხარეულო და სუფრის ტრადიციები}

ქართული ტრადიციული სამხარეულო 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`. Khmer 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]{1໐ 1໑ 1໒ 1໓ 1໔ 1໕} % 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 short 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 the other hand, with the Harfbuzz renderer diacritics are stacked correctly, but many hyphenation 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>	bg	Bulgarian <sup>ul</sup>
agq	Aghem	bm	Bambara
ak	Akan	bn	Bangla <sup>ul</sup>
am	Amharic <sup>ul</sup>	bo	Tibetan <sup>u</sup>
ar	Arabic <sup>ul</sup>	brx	Bodo
ar-DZ	Arabic <sup>ul</sup>	bs-Cyrl	Bosnian
ar-MA	Arabic <sup>ul</sup>	bs-Latn	Bosnian <sup>ul</sup>
ar-SY	Arabic <sup>ul</sup>	bs	Bosnian <sup>ul</sup>
as	Assamese	ca	Catalan <sup>ul</sup>
asa	Asu	ce	Chechen
ast	Asturian <sup>ul</sup>	cgg	Chiga
az-Cyrl	Azerbaijani	chr	Cherokee
az-Latn	Azerbaijani	ckb	Central Kurdish
az	Azerbaijani <sup>ul</sup>	cop	Coptic
bas	Basaa	cs	Czech <sup>ul</sup>
be	Belarusian <sup>ul</sup>	cu	Church Slavic
bem	Bemba	cu-Cyrs	Church Slavic
bez	Bena	cu-Glag	Church Slavic

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
en-NZ	English <sup>ul</sup>	ki	Kikuyu
en-US	English <sup>ul</sup>	kk	Kazakh
en	English <sup>ul</sup>	kkj	Kako
eo	Esperanto <sup>ul</sup>	kl	Kalaallisut
es-MX	Spanish <sup>ul</sup>	kln	Kalenjin
es	Spanish <sup>ul</sup>	km	Khmer
et	Estonian <sup>ul</sup>	kn	Kannada <sup>ul</sup>
eu	Basque <sup>ul</sup>	ko	Korean
ewo	Ewondo	kok	Konkani
fa	Persian <sup>ul</sup>	ks	Kashmiri
ff	Fulah	ksb	Shambala
fi	Finnish <sup>ul</sup>	ksf	Bafia
fil	Filipino	ksh	Colognian
fo	Faroese	kw	Cornish
fr	French <sup>ul</sup>	ky	Kyrgyz
fr-BE	French <sup>ul</sup>	lag	Langi
fr-CA	French <sup>ul</sup>	lb	Luxembourgish
fr-CH	French <sup>ul</sup>	lg	Ganda
fr-LU	French <sup>ul</sup>	lkt	Lakota
fur	Friulian <sup>ul</sup>	ln	Lingala
fy	Western Frisian	lo	Lao <sup>ul</sup>
ga	Irish <sup>ul</sup>	lrc	Northern Luri
gd	Scottish Gaelic <sup>ul</sup>	lt	Lithuanian <sup>ul</sup>
gl	Galician <sup>ul</sup>	lu	Luba-Katanga
grc	Ancient Greek <sup>ul</sup>	luo	Luo
gsw	Swiss German	luy	Luyia
gu	Gujarati	lv	Latvian <sup>ul</sup>
guz	Gusii	mas	Masai
gv	Manx	mer	Meru
ha-GH	Hausa	mfe	Morisyen
ha-NE	Hausa <sup>l</sup>	mg	Malagasy
ha	Hausa	mgh	Makhuwa-Meetto
haw	Hawaiian	mgo	Meta'
he	Hebrew <sup>ul</sup>	mk	Macedonian <sup>ul</sup>
hi	Hindi <sup>u</sup>	ml	Malayalam <sup>ul</sup>
hr	Croatian <sup>ul</sup>	mn	Mongolian

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>
om	Oromo	sw	Swahili
or	Odia	ta	Tamil <sup>u</sup>
os	Ossetic	te	Telugu <sup>ul</sup>
pa-Arab	Punjabi	teo	Teso
pa-Guru	Punjabi	th	Thai <sup>ul</sup>
pa	Punjabi	ti	Tigrinya
pl	Polish <sup>ul</sup>	tk	Turkmen <sup>ul</sup>
pms	Piedmontese <sup>ul</sup>	to	Tongan
ps	Pashto	tr	Turkish <sup>ul</sup>
pt-BR	Portuguese <sup>ul</sup>	twq	Tasawaq
pt-PT	Portuguese <sup>ul</sup>	tzm	Central Atlas Tamazight
pt	Portuguese <sup>ul</sup>	ug	Uyghur
qu	Quechua	uk	Ukrainian <sup>ul</sup>
rm	Romansh <sup>ul</sup>	ur	Urdu <sup>ul</sup>
rn	Rundi	uz-Arab	Uzbek
ro	Romanian <sup>ul</sup>	uz-Cyrl	Uzbek
rof	Rombo	uz-Latn	Uzbek
ru	Russian <sup>ul</sup>	uz	Uzbek
rw	Kinyarwanda	vai-Latn	Vai
rwk	Rwa	vai-Vaii	Vai
sa-Beng	Sanskrit	vai	Vai
sa-Deva	Sanskrit	vi	Vietnamese <sup>ul</sup>
sa-Gujr	Sanskrit	vun	Vunjo
sa-Knda	Sanskrit	wae	Walser
sa-Mlym	Sanskrit	xog	Soga
sa-Telu	Sanskrit	yav	Yangben
sa	Sanskrit	yi	Yiddish
sah	Sakha	yo	Yoruba
saq	Samburu	yue	Cantonese
sbp	Sangu	zgh	Standard Moroccan Tamazight
se	Northern Sami <sup>ul</sup>		
seh	Sena	zh-Hans-HK	Chinese
ses	Koyraboro Senni	zh-Hans-MO	Chinese
sg	Sango	zh-Hans-SG	Chinese
shi-Latn	Tachelhit	zh-Hans	Chinese
shi-Tfng	Tachelhit	zh-Hant-HK	Chinese

zh-Hant-MO	Chinese	zh	Chinese
zh-Hant	Chinese	zu	Zulu

---

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

---

aghem	burmese
akan	canadian
albanian	cantonese
american	catalan
amharic	centralatlastamazight
ancientgreek	centralkurdish
arabic	chechen
arabic-algeria	cherokee
arabic-DZ	chiga
arabic-morocco	chinese-hans-hk
arabic-MA	chinese-hans-mo
arabic-syria	chinese-hans-sg
arabic-SY	chinese-hans
armenian	chinese-hant-hk
assamese	chinese-hant-mo
asturian	chinese-hant
asu	chinese-simplified-hongkongsarchina
australian	chinese-simplified-macausarchina
austrian	chinese-simplified-singapore
azerbaijani-cyrillic	chinese-simplified
azerbaijani-cyrl	chinese-traditional-hongkongsarchina
azerbaijani-latin	chinese-traditional-macausarchina
azerbaijani-latn	chinese-traditional
azerbaijani	chinese
bafia	churchslavic
bambara	churchslavic-cyrs
basaa	churchslavic-oldcyrillic <sup>12</sup>
basque	churchsslavic-glag
belarusian	churchsslavic-glagolitic
bemba	cognian
bena	cornish
bengali	croatian
bodo	czech
bosnian-cyrillic	danish
bosnian-cyrl	duala
bosnian-latin	dutch
bosnian-latn	dzongkha
bosnian	embu
brazilian	english-au
breton	english-australia
british	english-ca
bulgarian	english-canada

---

<sup>12</sup>The name in the CLDR is Old Church Slavonic Cyrillic, but it has been shortened for practical reasons.



english-gb  
english-newzealand  
english-nz  
english-unitedkingdom  
english-unitedstates  
english-us  
english  
esperanto  
estonian  
ewe  
ewondo  
faroese  
filipino  
finnish  
french-be  
french-belgium  
french-ca  
french-canada  
french-ch  
french-lu  
french-luxembourg  
french-switzerland  
french  
friulian  
fulah  
galician  
ganda  
georgian  
german-at  
german-austria  
german-ch  
german-switzerland  
german  
greek  
gujarati  
gusii  
hausa-gh  
hausa-ghana  
hausa-ne  
hausa-niger  
hausa  
hawaiian  
hebrew  
hindi  
hungarian  
icelandic  
igbo  
inarisami  
indonesian  
interlingua  
irish  
italian  
japanese  
jolafonyi

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

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

sanskrit-telu  
sanskrit-telugu  
sanskrit  
scottishgaelic  
sena  
serbian-cyrillic-bosniaherzegovina  
serbian-cyrillic-kosovo  
serbian-cyrillic-montenegro  
serbian-cyrillic  
serbian-cyrl-ba  
serbian-cyrl-me  
serbian-cyrl-xk  
serbian-cyrl  
serbian-latin-bosniaherzegovina  
serbian-latin-kosovo  
serbian-latin-montenegro  
serbian-latin  
serbian-latn-ba  
serbian-latn-me  
serbian-latn-xk  
serbian-latn  
serbian  
shambala  
shona  
sichuanyi  
sinhala  
slovak  
slovene  
slovenian  
soga  
somali  
spanish-mexico  
spanish-mx  
spanish  
standardmoroccantamazight  
swahili  
swedish  
swissgerman  
tachelhit-latin  
tachelhit-latn  
tachelhit-tfng  
tachelhit-tifinagh  
tachelhit  
taita  
tamil  
tasawaq  
telugu  
teso  
thai  
tibetan  
tigrinya  
tongan  
turkish  
turkmen

ukenglish	vai-latn
ukrainian	vai-vai
uppersorbian	vai-vaii
urdu	vai
usenglish	vietnam
usorbian	vietnamese
uyghur	vunjo
uzbek-arab	walser
uzbek-arabic	welsh
uzbek-cyrillic	westernfrisian
uzbek-cyrl	yangben
uzbek-latin	yiddish
uzbek-latn	yoruba
uzbek	zarma
vai-latin	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>13</sup>

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

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

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

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

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

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

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

---

<sup>13</sup>See also the package `combofont` for a complementary approach.

**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}{עברית} 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`  $\{\langle\textit{language-name}\rangle\}\{\langle\textit{caption-name}\rangle\}\{\langle\textit{string}\rangle\}$

**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 imported 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\captionsenglish{%  
  \renewcommand\contentsname{Foo}%  
}
```

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

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

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

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

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

first loads danish.ldf, and then redefines the captions for danish (as provided by the ini file) and prevents hyphenation. The rest of the language definitions are not touched. 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=**  $\langle\text{language-tag}\rangle$

Loads only the strings. For example:

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

**hyphenrules=**  $\langle\text{language-list}\rangle$

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 T<sub>E</sub>X 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).

**New 3.58** Another special value is unhyphenated, which activates a line breking mode that allows spaces to be stretched to arbitrary amounts.

**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=**  $\langle\text{script-name}\rangle$

**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=**  $\langle\text{language-name}\rangle$

**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=**  $\langle\text{counter-name}\rangle$

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

**Alph=**  $\langle\text{counter-name}\rangle$

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=**  $\langle\text{base}\rangle$   $\langle\text{shrink}\rangle$   $\langle\text{stretch}\rangle$

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=**  $\langle\text{penalty}\rangle$

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).

**justification=** `kashida` | `elongated` | `unhyphenated`

**New 3.59** There are currently three options, mainly for the Arabic script. It sets the linebreaking and justification method, which can be based on the the ARABIC TATWEEL character or in the ‘justification alternatives’ OpenType table (`jalt`). For an explanation see the [babel site](#).

**linebreaking=** **New 3.59** Just a synonymous for `justification`.

**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 T<sub>E</sub>X 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):

- `\localenumerals{<style>}{<number>}`, like `\localenumerals{abjad}{15}`

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

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

The styles are:

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

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

## 1.18 Dates

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

`\localedate` [`<calendar=.., variant=..>`]{`<year>`}{`<month>`}{`<day>`}

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

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

## 1.19 Accessing language info

**\language** `\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** `{\language}{\true}{\false}`

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

**\localeinfo** `{\field}`

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

`name.english` as provided by the Unicode CLDR.

`tag.ini` is the tag of the ini file (the way this file is identified in its name).

`tag.bcp47` is the full BCP 47 tag (see the warning below).

`language.tag.bcp47` is the BCP 47 language tag.

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

`script.name`, as provided by the Unicode CLDR.

`script.tag.bcp47` is the BCP 47 tag of the script used by this locale.

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

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

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

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

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

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

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

**New 3.9a** It is customary to classify hyphens in two types: (1) *explicit* or *hard hyphens*, which in `TeX` are entered as `-`, and (2) *optional* or *soft hyphens*, which are entered as `\-`. Strictly, a *soft hyphen* is not a hyphen, but just a breaking opportunity or, in `TeX` terms, a “discretionary”; a *hard hyphen* is a hyphen with a breaking opportunity after it. A further type is a *non-breaking hyphen*, a hyphen without a breaking opportunity. In `TeX`, `-` and `\-` forbid further breaking opportunities in the word. This is the desired behavior very often, but not always, and therefore many languages provide shorthands for these cases. Unfortunately, this has not been done consistently: for example, `-` in Dutch, Portuguese, Catalan or Danish is a hard hyphen, while in German, Spanish, Norwegian, Slovak or Russian is a soft hyphen. Furthermore, some of them even redefine `\-`, so that you cannot insert a soft hyphen without breaking opportunities in the rest of the word. Therefore, some macros are provided with a set of basic “hyphens” which can be used by themselves, to define a user shorthand, or even in language files.

- `\babelhyphen{soft}` and `\babelhyphen{hard}` are self explanatory.
- `\babelhyphen{repeat}` inserts a hard hyphen which is repeated at the beginning of the next line, as done in languages like Polish, Portuguese and Spanish.
- `\babelhyphen{nobreak}` inserts a hard hyphen without a break after it (even if a space follows).
- `\babelhyphen{empty}` inserts a break opportunity without a hyphen at all.
- `\babelhyphen{<text>}` is a hard “hyphen” using `<text>` instead. A typical case is `\babelhyphen{/}`.

With all of them, hyphenation in the rest of the word is enabled. If you don’t want to enable it, there is a starred counterpart: `\babelhyphen*{soft}` (which in most cases is equivalent to the original `\-`), `\babelhyphen*{hard}`, etc.

Note `hard` is also good for isolated prefixes (eg, *anti-*) and `nobreak` for isolated suffixes (eg, *-ism*), but in both cases `\babelhyphen*{nobreak}` is usually better.

There are also some differences with `LaTeX`: (1) the character used is that set for the current font, while in `LaTeX` it is hardwired to `-` (a typical value); (2) the hyphen to be used in fonts with a negative `\hyphenchar` is `-`, like in `LaTeX`, but it can be changed to another value by redefining `\babenullhyphen`; (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}** {<language>} ... \end{hyphenrules}

The environment hyphenrules can be used to select *only* the hyphenation rules to be used (it can be used as command, too). This can for instance be used to select 'nohyphenation', provided that in language.dat the 'language' nohyphenation is defined by loading zerohyph.tex. It deactivates language shorthands, too (but not user shorthands). Except for these simple uses, hyphenrules is deprecated and other language\* (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** [*<language>*, *<language>*, ...]{*<patterns>*}

**New 3.9m** *In luatex only*,<sup>14</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

<sup>14</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.

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>15</sup>

It currently embraces `\babelprehyphenation` and `\babelposthyphenation`.

**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 this macro or the languages has been previously loaded as a class or package option, as the following example illustrates:

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

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

Arabic	<code>transliteration.dad</code>	Applies the transliteration system devised by Yannis Haralambous for dad (simple and T <sub>E</sub> X-friendly). Not yet complete, but sufficient for most texts.
Croatian	<code>digraphs.ligatures</code>	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.
Czech, Polish, Portuguese, Slovak, Spanish	<code>hyphen.repeat</code>	Explicit hyphens behave like <code>\babelhyphen{repeat}</code> .
Czech, Polish, Slovak	<code>oneletter.nobreak</code>	Converts a space after a non-syllabic preposition or conjunction into a non-breaking space.
Greek	<code>diaeresis.hyphen</code>	Removes the diaeresis above iota and upsilon if hyphenated just before. It works with the three variants.
Hindi, Sanskrit	<code>transliteration.hk</code>	The Harvard-Kyoto system to romanize Devanagari.
Hindi, Sanskrit	<code>punctuation.space</code>	Inserts a space before the following four characters: <i>! ? ; .</i>
Hungarian	<code>digraphs.hyphen</code>	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.

<sup>15</sup>They are similar in concept, but not the same, as those in Unicode.

Indic scripts	danda.nobreak	Prevents a line break before a danda or double danda if there is a space. For Assamese, Bengali, Gujarati, Hindi, Kannada, Malayalam, Marathi, Oriya, Tamil, Telugu.
Arabic, Persian	kashida.plain	Experimental. A very simple and basic transform for ‘plain’ Arabic fonts, which attempts to distribute the tatwil as evenly as possible (starting at the end of the line). See the news for version 3.59.
Serbian	transliteration.gajica	(Note serbian with ini files refers to the Cyrillic script, which is here the target.) The standard system devised by Ljudevit Gaj.

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

**New 3.37-3.39** With *luatex* it is 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. A few rules are currently provided (see above), 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  $([\text{t}\acute{u}])$ , the replacement could be  $\{1|\text{t}\acute{u}|\text{t}\acute{u}\}$ , which maps  $\text{t}\acute{u}$  to  $\text{t}\acute{u}$ , 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**  $\{\langle locale-name \rangle\}\{\langle lua-pattern \rangle\}\{\langle replacement \rangle\}$

**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, discretionaries are not accepted. 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  $\text{ž}$  as zh and  $\text{š}$  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
}

```

**NOTE** With luatex there is another approach to make text transformations, with the function `fonts.handlers.otf.addfeature`, which adds new features to an OTF font (substitution and positioning). These features can be made language-dependent, and babel by default recognizes this setting if the font has been declared with `\babelfont`. The *transforms* mechanism supplements rather than replaces OTF features.

With xetex, where *transforms* are not available, there is still another approach, with font mappings, mainly meant to perform encoding conversions and transliterations. Mappings, however, are linked to fonts, not to languages.

## 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{
  autoloader.bcp47 = on,
  autoloader.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>16</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>17</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>16</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>17</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 الهيليني (الاجريقي) بـ
    Arabia أو Aravia (بالاغريقية (Αραβία), استخدم الرومان ثلاث
    بادئات بـ“Arabia” على ثلاث مناطق من شبه الجزيرة العربية، إلا أنها
    حقيقةً كانت أكبر مما تعرف عليه اليوم.

\end{document}
```

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

```
\documentclass{book}

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

\babelprovide[onchar=ids fonts]{arabic}

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

\begin{document}

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

\end{document}
```

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

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

```
\newcommand\refrange[2]{\babelsublr{\textthe{\ref{#1}}}-\textthe{\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>18</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** .

**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:

<sup>18</sup>Next on the roadmap are counters and numeral systems in general. Expect some minor readjustments.

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

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

Digits in pdfTeX must be marked up explicitly (unlike LaTeX with `bidi=basic` or `bidi=basic-r` and, usually, `xetex`). This command is provided to set  $\{\langle lr\text{-}text\rangle\}$  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**  $\{\langle section\text{-}name\rangle\}$

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**  $\{\langle cmd\rangle\}\{\langle local\text{-}language\rangle\}\{\langle before\rangle\}\{\langle after\rangle\}$

**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}\{()\}
```

(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 with a certain `\name` 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  $\text{\TeX}$  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>19</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.

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  $\langle file \rangle$ , which creates  $\langle file \rangle$ .tex; you can then typeset the latter with  $\LaTeX$ .

---

<sup>19</sup>The two last name comes from the times when they had to be shortened to 8 characters

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

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

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

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

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

## 1.29 Tweaking some features

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

**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`, `justify.arabic`. For example, you can set `\babeladjust{bidi.text=off}` if you are using an alternative algorithm or with large sections not requiring it. 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 T<sub>E</sub>X only takes into account the values when the paragraph is hyphenated, i.e., when it has been finished.<sup>20</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 T<sub>E</sub>X, 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 a similar issue with floats, too. 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 T<sub>E</sub>X enter in an infinite loop in some rare cases. (Another issue in the 'to do' list, although there is a partial solution.)

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

**csquotes** Logical markup for quotes.

**iflang** Tests correctly the current language.

**hyphsubst** Selects a different set of patterns for a language.

**translator** An open platform for packages that need to be localized.

**siunitx** Typesetting of numbers and physical quantities.

**biblatex** Programmable bibliographies and citations.

**bicaption** Bilingual captions.

**babelbib** Multilingual bibliographies.

**microtype** Adjusts the typesetting according to some languages (kerning and spacing).  
Ligatures can be disabled.

**substitutefont** Combines fonts in several encodings.

**mkpattern** Generates hyphenation patterns.

**tracklang** Tracks which languages have been requested.

**ucharclasses** (xetex) Switches fonts when you switch from one Unicode block to another.

**zhspacing** Spacing for CJK documents in xetex.

### 1.31 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>21</sup> But that is the easy part, because they don't require modifying the L<sup>A</sup>T<sub>E</sub>X internals. Calendars (Arabic, Persian, Indic, etc.) are under study.

<sup>20</sup>This explains why L<sup>A</sup>T<sub>E</sub>X assumes the lowercase mapping of T1 and does not provide a tool for multiple mappings. Unfortunately, `\savingsphcodes` is not a solution either, because lccodes for hyphenation are frozen in the format and cannot be changed.

<sup>21</sup>See for example POSIX, ISO 14652 and the Unicode Common Locale Data Repository (CLDR). Those systems, however, have limited application to T<sub>E</sub>X because their aim is just to display information and not fine typesetting.

Also interesting are differences in the sentence structure or related to it. For example, in Basque the number precedes the name (including chapters), in Hungarian “from (1)” is “(1)-ből”, but “from (3)” is “(3)-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).

**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>22</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>23</sup>

### 2.1 Format

In that file the person who maintains a  $\TeX$  environment has to record for which languages he has hyphenation patterns *and* in which files these are stored<sup>24</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  $\LaTeX$  that the hyphenation patterns just processed have to be known under an alternative name. Here is an example:

<sup>22</sup>This feature was added to 3.9o, but it was buggy. Both 3.9o and 3.9p are deprecated.

<sup>23</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>24</sup>This is because different operating systems sometimes use very different file-naming conventions.

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

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

You may also set the font encoding the patterns are intended for by following the language name by a colon and the encoding code.<sup>25</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.

### 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

<sup>25</sup>This is not a new feature, but in former versions it didn't work correctly.

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

- When a language definition file is loaded, it can define `\l@⟨lang⟩` to be a dialect of `\language0` when `\l@⟨lang⟩` is undefined.
- Language names must be all lowercase. If an unknown language is selected, babel will attempt setting it after lowercasing its name.
- The semantics of modifiers is not defined (on purpose). In most cases, they will just be simple separated options (eg, spanish), but a language might require, say, a set of options organized as a tree with suboptions (in such a case, the recommended separator is `/`).

Some recommendations:

- The preferred shorthand is `"`, which is not used in  $\LaTeX$  (quotes are entered as ``` and `'`). Other good choices are characters which are not used in a certain context (eg, `=` in an ancient language). Note however `=`, `<`, `>`, `:` and the like can be dangerous, because they may be used as part of the syntax of some elements (numeric expressions, key/value pairs, etc.).
- Captions should not contain shorthands or encoding-dependent commands (the latter is not always possible, but should be clearly documented). They should be defined using the LICR. You may also use the new tools for encoded strings, described below.
- Avoid adding things to `\noextras⟨lang⟩` except for `umlauthigh` and friends, `\bbl@deactivate`, `\bbl@(non) frenchspacing`, and language-specific macros. Use always, if possible, `\bbl@save` and `\bbl@savevariable` (except if you still want to have access to the previous value). Do not reset a macro or a setting to a hardcoded value. Never. Instead save its value in `\extras⟨lang⟩`.
- Do not switch scripts. If you want to make sure a set of glyphs is used, switch either the font encoding (low-level) or the language (high-level, which in turn may switch the font encoding). Usage of things like `\latintext` is deprecated.<sup>26</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.

<sup>26</sup>But not removed, for backward compatibility.

- 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, ot f, 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://latex3.github.io/babel/guides/list-of-locale-templates.html>.

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

### 3.2 Basic macros

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

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

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

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

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

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

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

**\captions<lang>** The macro \captions<lang> defines the macros that hold the texts to replace the original hard-wired texts.

**\date<lang>** The macro \date<lang> defines \today.

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

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



<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 $\TeX$ 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, $\TeX$ 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 $\TeX$ to use a font from the second family when a font from the first family in the given encoding seems to be needed.

### 3.3 Skeleton

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

```

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

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

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

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

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

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

\StartBabelCommands*{<language>}{date}

```



```

\SetString\monthiname{<name of first month>}
% More strings

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

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

\EndBabelCommands

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

\ldf@finish{<language>}

```

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

```

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

```

### 3.4 Support for active characters

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

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

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

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

`\bbl@add@special` The  $\text{\TeX}$ book states: “Plain  $\text{\TeX}$  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`.  $\text{\LaTeX}$  adds another macro called `\@sanitize` representing the same character set, but without the curly braces. The macros `\bbl@add@special<char>` and `\bbl@remove@special<char>` add and remove the character `<char>` to these two sets.

### 3.5 Support for saving macro definitions

Language definition files may want to 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>27</sup>.

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

**`\babel@savevariable`** A second macro is provided to save the current value of a variable. In this context, anything that is allowed after the `\` the primitive is considered to be a variable. The macro takes one argument, the `<variable>`.  
The effect of the preceding macros is to append a piece of code to the current definition of `\originalTeX`. When `\originalTeX` is expanded, this code restores the previous definition of the control sequence or the previous value of the variable.

### 3.6 Support for extending macros

**`\addto`** The macro `\addto{<control sequence>}{<TeX code>}` can be used to extend the definition of a macro. The macro need not be defined (ie, it can be undefined or `\relax`). This macro can, for instance, be used in adding instructions to a macro like `\extrasenglish`. Be careful when using this macro, because depending on the case the assignment can be either global (usually) or local (sometimes). That does not seem very consistent, but this behavior is preserved for backward compatibility. If you are using `etoolbox`, by Philipp Lehman, consider using the tools provided by this package instead of `\addto`.

### 3.7 Macros common to a number of languages

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

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

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

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

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

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

### 3.8 Encoding-dependent strings

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

Furthermore, it must be activated explicitly, with the package option `strings`. If there is no `strings`, these blocks are ignored, except `\SetCases` (and except if forced as described

<sup>27</sup>This mechanism was introduced by Bernd Raichle.

below). In other words, the old way of defining/switching strings still works and it's used by default.

It consists of 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  $\langle\textit{category}\rangle$  is either `captions`, `date` or `extras`. You must stick to these three categories, even if no error is raised when using other name.<sup>28</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:

---

<sup>28</sup>In future releases further categories may be added.

```

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

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

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

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


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

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

\EndBabelCommands

```

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

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

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>29</sup>

$\backslash EndBabelCommands$  Marks the end of the series of blocks.

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

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

<sup>29</sup>This replaces in 3.9g a short-lived  $\backslash UseStrings$  which has been removed because it did not work.

**\SetString** {*<macro-name>*}{*<string>*}

Adds *<macro-name>* to the current category, and defines globally *<lang-macro-name>* to *<code>* (after applying the transformation corresponding to the current charset or defined with the hook `stringprocess`).

Use this command to define strings, without including any “logic” if possible, which should be a separated macro. See the example above for the date.

**\SetStringLoop** {*<macro-name>*}{*<string-list>*}

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

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

#1 is replaced by the roman numeral.

**\SetCase** [*<map-list>*]{*<toupper-code>*}{*<tolower-code>*}

Sets globally code to be executed at `\MakeUppercase` and `\MakeLowercase`. The code would typically be things like `\let\BB\bb` and `\uccode` or `\lccode` (although for the reasons explained above, changes in lc/uc codes may not work). A *<map-list>* is a series of macros using the internal format of `\@uclclist` (eg, `\bb\BB\cc\CC`). The mandatory arguments take precedence over the optional one. This command, unlike `\SetString`, is executed always (even without strings), and it is intended for minor readjustments only. For example, as T1 is the default case mapping in  $\TeX$ , 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=`I\relax
 \uccode`1=`I\relax}
{\lccode`I=`i\relax
 \lccode`I=`1\relax}

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

\EndBabelCommands
```

(Note the mapping for OT1 is not complete.)

**\SetHyphenMap** {*<to-lower-macros>*}

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

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

- `\BabelLower{⟨ucode⟩}{⟨lcode⟩}` is similar to `\lcode` but it's ignored if the char has been set and saves the original lcode to restore it when switching the language (except with `hyphenmap=first`).
- `\BabelLowerMM{⟨ucode-from⟩}{⟨ucode-to⟩}{⟨step⟩}{⟨lcode-from⟩}` loops through the given uppercase codes, using the step, and assigns them the lcode, which is also increased (MM stands for *many-to-many*).
- `\BabelLowerMO{⟨ucode-from⟩}{⟨ucode-to⟩}{⟨step⟩}{⟨lcode⟩}` loops through the given uppercase codes, using the step, and assigns them the lcode, 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 an uppercase letter. It can be just a letter (eg, babel.name.A, babel.name.B) or a name (eg, date.long.Nominative, date.long.Formal, but no language is currently using the latter). *Multi-letter* qualifiers are forward compatible in the sense they won’t conflict with new “global” keys (which start always with a lowercase case). There is an exception, however: the section counter s has been devised to have arbitrary keys, so you can add lowercased keys if you want.

## 7 Tools

```
1 <<version=3.63.2485>>
2 <<date=2021/09/04>>
```

**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` `\bbl@afterfi` Because the code that is used in the handling of active characters may need to look ahead, we take extra care to ‘throw’ it over the `\else` and `\fi` parts of an `\if`-statement<sup>30</sup>. 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   \let\<\bbl@exp@en
33   \let\[\bbl@exp@ue
34   \edef\bbl@exp@aux{\endgroup#1}%
35 }
```

<sup>30</sup>This code is based on code presented in TUGboat vol. 12, no2, June 1991 in “An expansion Power Lemma” by Sonja Maus.



```

35 \bbl@exp@aux}
36 \def\bbl@exp@en#1>{\expandafter\noexpand\csname#1\endcsname}%
37 \def\bbl@exp@ue#1]{%
38 \unexpanded\expandafter\expandafter\expandafter{\csname#1\endcsname}}%

```

\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.

```

39 \def\bbl@tempa#1{%
40 \long\def\bbl@trim##1##2{%
41 \futurelet\bbl@trim@a\bbl@trim@c##2\@nil\@nil#1\@nil\relax{##1}}%
42 \def\bbl@trim@c{%
43 \ifx\bbl@trim@a\@sptoken
44 \expandafter\bbl@trim@b
45 \else
46 \expandafter\bbl@trim@b\expandafter#1%
47 \fi}%
48 \long\def\bbl@trim@b#1##1 \@nil{\bbl@trim@i##1}}
49 \bbl@tempa{ }
50 \long\def\bbl@trim@i#1\@nil#2\relax#3{#3{#1}}
51 \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 does not waste memory.

```

52 \begingroup
53 \gdef\bbl@ifunset#1{%
54 \expandafter\ifx\csname#1\endcsname\relax
55 \expandafter\@firstoftwo
56 \else
57 \expandafter\@secondoftwo
58 \fi}
59 \bbl@ifunset{ifcsname}% TODO. A better test?
60 {}%
61 {\gdef\bbl@ifunset#1{%
62 \ifcsname#1\endcsname
63 \expandafter\ifx\csname#1\endcsname\relax
64 \bbl@afterelse\expandafter\@firstoftwo
65 \else
66 \bbl@afterfi\expandafter\@secondoftwo
67 \fi
68 \else
69 \expandafter\@firstoftwo
70 \fi}}
71 \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,

```

72 \def\bbl@ifblank#1{%
73 \bbl@ifblank@i#1\@nil\@nil\@secondoftwo\@firstoftwo\@nil}
74 \long\def\bbl@ifblank@i#1#2\@nil#3#4#5\@nil{#4}
75 \def\bbl@ifset#1#2#3{%
76 \bbl@ifunset{#1}{#3}{\bbl@exp{\bbl@ifblank{#1}}{#3}{#2}}}

```

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

```

77 \def\bbl@forkv#1#2{%
78   \def\bbl@kvcmd##1##2##3{#2}%
79   \bbl@kvnext#1,\@nil,}
80 \def\bbl@kvnext#1,{%
81   \ifx\@nil#1\relax\else
82     \bbl@ifblank{#1}{\bbl@forkv@eq#1=\@empty=\@nil{#1}}%
83     \expandafter\bbl@kvnext
84   \fi}
85 \def\bbl@forkv@eq#1=#2=#3\@nil#4{%
86   \bbl@trim@def\bbl@forkv@a{#1}%
87   \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).

```

88 \def\bbl@vforeach#1#2{%
89   \def\bbl@forcmd##1{#2}%
90   \bbl@fornext#1,\@nil,}
91 \def\bbl@fornext#1,{%
92   \ifx\@nil#1\relax\else
93     \bbl@ifblank{#1}{\bbl@trim\bbl@forcmd{#1}}%
94     \expandafter\bbl@fornext
95   \fi}
96 \def\bbl@foreach#1{\expandafter\bbl@vforeach\expandafter{#1}}

```

`\bbl@replace` Returns implicitly `\toks@` with the modified string.

```

97 \def\bbl@replace#1#2#3{% in #1 -> repl #2 by #3
98   \toks@{}%
99   \def\bbl@replace@aux##1#2##2#2{%
100     \ifx\bbl@nil##2%
101       \toks@\expandafter{\the\toks@##1}%
102     \else
103       \toks@\expandafter{\the\toks@##1#3}%
104       \bbl@afterfi
105       \bbl@replace@aux##2#2%
106     \fi}%
107   \expandafter\bbl@replace@aux#1#2\bbl@nil#2%
108   \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).

```

109 \ifx\detokenize\@undefined\else % Unused macros if old Plain TeX
110   \bbl@exp{\def\\bbl@parsedef##1\detokenize{macro:}}#2->#3\relax{%
111     \def\bbl@tempa{#1}%
112     \def\bbl@tempb{#2}%
113     \def\bbl@tempe{#3}}
114   \def\bbl@sreplace#1#2#3{%
115     \begingroup
116     \expandafter\bbl@parsedef\meaning#1\relax
117     \def\bbl@tempc{#2}%
118     \edef\bbl@tempc{\expandafter\strip@prefix\meaning\bbl@tempc}%
119     \def\bbl@tempd{#3}%
120     \edef\bbl@tempd{\expandafter\strip@prefix\meaning\bbl@tempd}%
121     \bbl@xin{\bbl@tempc}{\bbl@tempe}% If not in macro, do nothing
122     \ifin@
123       \bbl@exp{\bbl@replace\\bbl@tempe{\bbl@tempc}{\bbl@tempd}}%
124       \def\bbl@tempc{% Expanded an executed below as 'uplevel'

```

```

125         \\makeatletter % "internal" macros with @ are assumed
126         \\scantokens{%
127             \bbl@tempa\\@namedef{\bbl@stripslash#1}\bbl@tempb{\bbl@tempe}}%
128             \catcode64=\the\catcode64\relax}% Restore @
129     \else
130         \let\bbl@tempc\@empty % Not \relax
131     \fi
132     \bbl@exp{%      For the 'uplevel' assignments
133 \endgroup
134     \bbl@tempc}} % empty or expand to set #1 with changes
135 \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 in your language style if you want.

```

136 \def\bbl@ifsamestring#1#2{%
137 \begingroup
138 \protected@edef\bbl@tempb{#1}%
139 \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
140 \protected@edef\bbl@tempc{#2}%
141 \edef\bbl@tempc{\expandafter\strip@prefix\meaning\bbl@tempc}%
142 \ifx\bbl@tempb\bbl@tempc
143     \aftergroup\@firstoftwo
144 \else
145     \aftergroup\@secondoftwo
146 \fi
147 \endgroup}
148 \chardef\bbl@engine=%
149 \ifx\directlua\@undefined
150     \ifx\XeTeXinputencoding\@undefined
151         \z@
152     \else
153         \tw@
154     \fi
155 \else
156     \@ne
157 \fi

```

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

```

158 \def\bbl@bsphack{%
159 \ifhmode
160     \hskip\z@skip
161     \def\bbl@esphack{\loop\ifdim\lastskip>\z@\unskip\repeat\unskip}%
162 \else
163     \let\bbl@esphack\@empty
164 \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`.

```

165 \def\bbl@cased{%
166 \ifx\oe\OE
167     \expandafter\in@\expandafter
168     {\expandafter\OE\expandafter}\expandafter{\oe}%
169 \ifin@
170     \bbl@afterelse\expandafter\MakeUppercase
171 \else
172     \bbl@afterfi\expandafter\MakeLowercase
173 \fi
174 \else

```

```

175 \expandafter\@firstofone
176 \fi}

```

An alternative to `\IfFormatAtLeastTF` for old versions. Temporary.

```

177 \ifx\IfFormatAtLeastTF\undefined
178 \def\bbl@ifformatlater{\@ifl@t@r\fmtversion}
179 \else
180 \let\bbl@ifformatlater\IfFormatAtLeastTF
181 \fi

```

The following adds some code to `\extras...` both before and after, while avoiding doing it twice. It's somewhat convoluted, to deal with `#`'s. Used to deal with `alph`, `Alph` and `frenchspacing` when there are already changes (with `\babel@save`).

```

182 \def\bbl@extras@wrap#1#2#3{% 1:in-test, 2:before, 3:after
183 \toks@{\expandafter\expandafter\expandafter{%
184 \csname extras\language\endcsname}%
185 \bbl@exp{\in@{#1}{\the\toks@}}}%
186 \ifin@ \else
187 \@temptokena{#2}%
188 \edef\bbl@tempc{\the\@temptokena\the\toks@}%
189 \toks@\expandafter{\bbl@tempc#3}%
190 \expandafter\edef\csname extras\language\endcsname{\the\toks@}%
191 \fi}
192 <</Basic macros>>

```

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

```

193 <<*Make sure ProvidesFile is defined>> ≡
194 \ifx\ProvidesFile\undefined
195 \def\ProvidesFile#1[#2 #3 #4]{%
196 \wlog{File: #1 #4 #3 <#2>}%
197 \let\ProvidesFile\undefined}
198 \fi
199 <</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 require loading `switch.def` in the format.

```

200 <<*Define core switching macros>> ≡
201 \ifx\language\undefined
202 \csname newcount\endcsname\language
203 \fi
204 <</Define core switching macros>>

```

`\last@language` Another counter is used to keep track of the allocated languages.  $\TeX$  and  $\LaTeX$  reserves for this purpose the count 19.

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

```

205 <<*Define core switching macros>> ≡
206 \countdef\last@language=19
207 \def\addlanguage{\csname newlanguage\endcsname}
208 <</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. 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 (~~La~~T<sub>E</sub>X, `babel.sty`)

```

209 <*package>
210 \NeedsTeXFormat{LaTeX2e}[2005/12/01]
211 \ProvidesPackage{babel}[\<<date>> \<<version>>] The Babel package]

Start with some “private” debugging tool, and then define macros for errors.
212 \@ifpackagewith{babel}{debug}
213   {\providecommand\bbl@trace[1]{\message{^^J[ #1 ]}}%
214    \let\bbl@debug\@firstofone
215    \ifx\directlua\@undefined\else
216      \directlua{ Babel = Babel or {}
217        Babel.debug = true }%
218      \input{babel-debug.tex}%
219    \fi}
220 {\providecommand\bbl@trace[1]{}%
221  \let\bbl@debug\@gobble
222  \ifx\directlua\@undefined\else
223    \directlua{ Babel = Babel or {}
224      Babel.debug = false }%
225  \fi}
226 \def\bbl@error#1#2{%
227   \begingroup
228     \def\{\MessageBreak}%
229     \PackageError{babel}{#1}{#2}%
230   \endgroup}
231 \def\bbl@warning#1{%
232   \begingroup
233     \def\{\MessageBreak}%
234     \PackageWarning{babel}{#1}%
235   \endgroup}
236 \def\bbl@infowarn#1{%
237   \begingroup
238     \def\{\MessageBreak}%
239     \GenericWarning
240       {(babel) \spaces\@spaces\@spaces}%
241       {Package babel Info: #1}%
242   \endgroup}
243 \def\bbl@info#1{%
244   \begingroup
245     \def\{\MessageBreak}%
246     \PackageInfo{babel}{#1}%
247   \endgroup}

```

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.

But first, include here the *Basic macros* defined above.

```

248 <<Basic macros>>
249 \@ifpackagewith{babel}{silent}
250   {\let\bbl@info\@gobble
251    \let\bbl@infowarn\@gobble

```

```

252 \let\bbl@warning\@gobble}
253 {}
254 %
255 \def\AfterBabelLanguage#1{%
256 \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.
257 \ifx\bbl@languages\@undefined\else
258 \begingroup
259 \catcode\^^I=12
260 \@ifpackagewith{babel}{showlanguages}{%
261 \begingroup
262 \def\bbl@elt#1#2#3#4{\wlog{#2^^I#1^^I#3^^I#4}}%
263 \wlog{<*languages>}%
264 \bbl@languages
265 \wlog{</languages>}%
266 \endgroup}{%
267 \endgroup
268 \def\bbl@elt#1#2#3#4{%
269 \ifnum#2=\z@
270 \gdef\bbl@nulllanguage{#1}%
271 \def\bbl@elt##1##2##3##4{}}%
272 \fi}%
273 \bbl@languages
274 \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`.

```

275 \bbl@trace{Defining option 'base'}
276 \@ifpackagewith{babel}{base}{%
277 \let\bbl@onlyswitch\@empty
278 \let\bbl@provide@locale\relax
279 \input babel.def
280 \let\bbl@onlyswitch\@undefined
281 \ifx\directlua\@undefined
282 \DeclareOption*{\bbl@patterns{\CurrentOption}}%
283 \else
284 \input luababel.def
285 \DeclareOption*{\bbl@patterns@lua{\CurrentOption}}%
286 \fi
287 \DeclareOption{base}{}%
288 \DeclareOption{showlanguages}{}%
289 \ProcessOptions
290 \global\expandafter\let\csname opt@babel.sty\endcsname\relax
291 \global\expandafter\let\csname ver@babel.sty\endcsname\relax
292 \global\let\@ifl@ter@\@ifl@ter
293 \def\@ifl@ter#1#2#3#4#5{\global\let\@ifl@ter\@ifl@ter@}%
294 \endinput}{}%

```

### 7.4 key=value options and other general option

The following macros extract language modifiers, and only real package options are kept in the option list. Modifiers are saved and assigned to `\BabelModifiers` at `\bbl@load@language`; when no

modifiers have been given, the former is `\relax`. How modifiers are handled are left to language styles; they can use `\in@`, loop them with `\for` or load `keyval`, for example.

```

295 \bbl@trace{key=value and another general options}
296 \bbl@csarg\let{\tempa\expandafter}\csname opt@babel.sty\endcsname
297 \def\bbl@tempb#1.#2{% Remove trailing dot
298   #1\ifx\@empty#2\else,\bbl@afterfi\bbl@tempb#2\fi}%
299 \def\bbl@tempd#1.#2\@nnil{% TODO. Refactor lists?
300   \ifx\@empty#2%
301     \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
302   \else
303     \in@{,provide=}{, #1}%
304     \ifin@
305       \edef\bbl@tempc{%
306         \ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.\bbl@tempb#2}%
307     \else
308       \in@{=}{#1}%
309       \ifin@
310         \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1.#2}%
311       \else
312         \edef\bbl@tempc{\ifx\bbl@tempc\@empty\else\bbl@tempc,\fi#1}%
313         \bbl@csarg\edef{mod@#1}{\bbl@tempb#2}%
314       \fi
315     \fi
316   \fi}
317 \let\bbl@tempc\@empty
318 \bbl@foreach\bbl@tempa{\bbl@tempd#1.\@empty\@nnil}
319 \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.

```

320 \DeclareOption{KeepShorthandsActive}{}
321 \DeclareOption{activeacute}{}
322 \DeclareOption{activegrave}{}
323 \DeclareOption{debug}{}
324 \DeclareOption{noconfigs}{}
325 \DeclareOption{showlanguages}{}
326 \DeclareOption{silent}{}
327 % \DeclareOption{mono}{}
328 \DeclareOption{shorthands=off}{\bbl@tempa shorthands=\bbl@tempa}
329 \chardef\bbl@iniflag\z@
330 \DeclareOption{provide=*}{\chardef\bbl@iniflag\@ne} % main -> +1
331 \DeclareOption{provide+=*}{\chardef\bbl@iniflag\tw@} % add = 2
332 \DeclareOption{provide+=*}{\chardef\bbl@iniflag\thr@@} % add + main
333 % A separate option
334 \let\bbl@autoload@options\@empty
335 \DeclareOption{provide=@*}{\def\bbl@autoload@options{import}}
336 % Don't use. Experimental. TODO.
337 \newif\ifbbl@single
338 \DeclareOption{selectors=off}{\bbl@singletrue}
339 <<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.

```

340 \let\bbl@opt@shorthands\@nnil
341 \let\bbl@opt@config\@nnil
342 \let\bbl@opt@main\@nnil

```

```

343 \let\bbl@opt@headfoot\@nnil
344 \let\bbl@opt@layout\@nnil
345 \let\bbl@opt@provide\@nnil

```

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

```

346 \def\bbl@tempa#1=#2\bbl@tempa{%
347   \bbl@csarg\ifx{opt@#1}\@nnil
348     \bbl@csarg\edef{opt@#1}{#2}%
349   \else
350     \bbl@error
351     {Bad option '#1=#2'. Either you have misspelled the\\%
352       key or there is a previous setting of '#1'. Valid\\%
353       keys are, among others, 'shorthands', 'main', 'bidi',\\%
354       'strings', 'config', 'headfoot', 'safe', 'math'.}%
355     {See the manual for further details.}
356   \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.

```

357 \let\bbl@language@opts\@empty
358 \DeclareOption*{%
359   \bbl@xin@{\string=}{\CurrentOption}%
360   \ifin@
361     \expandafter\bbl@tempa\CurrentOption\bbl@tempa
362   \else
363     \bbl@xin@{,\CurrentOption,}{,\bbl@language@opts,%
364     \ifin@
365       \bbl@exp{\\bbl@replace\\bbl@language@opts{,\CurrentOption,}{}}%
366     \fi
367     \edef\bbl@language@opts{\bbl@language@opts,\CurrentOption,}
368   \fi}

```

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

```

369 \ProcessOptions*
370 \ifx\bbl@opt@provide\@nnil
371   \let\bbl@opt@provide\@empty %%%% MOVE above
372 \else
373   \chardef\bbl@iniflag\@ne
374   \bbl@exp{\\bbl@forkv{\@nameuse{@raw@opt@babel.sty}}}{%
375     \in@{,provide,}{,#1,}%
376     \ifin@
377       \def\bbl@opt@provide{#2}%
378       \bbl@replace\bbl@opt@provide{;}{,}%
379     \fi}
380 \fi
381 %

```

## 7.5 Conditional loading of shorthands

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

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

```

382 \bbl@trace{Conditional loading of shorthands}
383 \def\bbl@sh@string#1{%
384   \ifx#1\@empty\else
385     \ifx#1t\string~%

```



```

386 \else\ifx#1c\string,%
387 \else\string#1%
388 \fi\fi
389 \expandafter\bbbl@sh@string
390 \fi}
391 \ifx\bbbl@opt@shorthands\@nnil
392 \def\bbbl@ifshorthand#1#2#3{#2}%
393 \else\ifx\bbbl@opt@shorthands\@empty
394 \def\bbbl@ifshorthand#1#2#3{#3}%
395 \else

```

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

```

396 \def\bbbl@ifshorthand#1{%
397 \bbbl@xin@{\string#1}{\bbbl@opt@shorthands}%
398 \ifin@
399 \expandafter\@firstoftwo
400 \else
401 \expandafter\@secondoftwo
402 \fi}

```

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

```

403 \edef\bbbl@opt@shorthands{%
404 \expandafter\bbbl@sh@string\bbbl@opt@shorthands\@empty}%

```

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

```

405 \bbbl@ifshorthand{'}%
406 {\PassOptionsToPackage{activeacute}{babel}}{}
407 \bbbl@ifshorthand{`}%
408 {\PassOptionsToPackage{activegrave}{babel}}{}
409 \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.

```

410 \ifx\bbbl@opt@headfoot\@nnil\else
411 \g@addto@macro\@resetactivechars{%
412 \set@typeset@protect
413 \expandafter\select@language@x\expandafter{\bbbl@opt@headfoot}%
414 \let\protect\noexpand}
415 \fi

```

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

```

416 \ifx\bbbl@opt@safe\@undefined
417 \def\bbbl@opt@safe{BR}
418 \fi

```

Make sure the language set with ‘main’ is the last one.

```

419 \ifx\bbbl@opt@main\@nnil\else
420 \edef\bbbl@language@opts{\bbbl@language@opts,\bbbl@opt@main,}
421 \fi

```

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

```

422 \bbbl@trace{Defining IfBabelLayout}
423 \ifx\bbbl@opt@layout\@nnil
424 \newcommand\IfBabelLayout[3]{#3}%
425 \else
426 \newcommand\IfBabelLayout[1]{%

```

```

427 \expandafter\in@{.#1.}{.\bbl@opt@layout.}%
428 \ifin@
429 \expandafter\@firstoftwo
430 \else
431 \expandafter\@secondoftwo
432 \fi}
433 \fi
434 \endpackage
435 \core

```

## 7.6 Interlude for Plain

Because of the way docstrip works, we need to insert some code for Plain here. However, the tools provided by the babel installer for literate programming makes this section a short interlude, because the actual code is below, tagged as *Emulate LaTeX*.

```

436 \ifx\ldf@quit\undefined\else
437 \endinput\fi % Same line!
438 <<Make sure ProvidesFile is defined>>
439 \ProvidesFile{babel.def}[\<date>] \<version>] Babel common definitions]
440 \ifx\AtBeginDocument\undefined % TODO. change test.
441 <<Emulate LaTeX>>
442 \fi

```

That is all for the moment. Now follows some common stuff, for both Plain and  $\text{\LaTeX}$ . After it, we will resume the  $\text{\LaTeX}$ -only stuff.

```

443 \endcore
444 \package | core

```

## 8 Multiple languages

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

Plain  $\text{\TeX}$  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.

```

445 \def\bbl@version{\<version>}
446 \def\bbl@date{\<date>}
447 <<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.

```

448 \def\adddialect#1#2{%
449 \global\chardef#1#2\relax
450 \bbl@usehooks{adddialect}{\#1}{\#2}}%
451 \begingroup
452 \count@#1\relax
453 \def\bbl@elt##1##2##3##4{%
454 \ifnum\count@=##2\relax
455 \edef\bbl@tempa{\expandafter\@gobbletwo\string#1}%
456 \bbl@info{Hyphen rules for '\expandafter\@gobble\bbl@tempa'
457 set to \expandafter\string\csname l@##1\endcsname\%
458 (\string\language\the\count@). Reported}%
459 \def\bbl@elt####1####2####3####4}%
460 \fi}%
461 \bbl@cs{languages}%
462 \endgroup

```

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

The argument of `\bbl@fixname` has to be a macro name, as it may get “fixed” if casing (lc/uc) is wrong. It’s an attempt 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.

```

463 \def\bbl@fixname#1{%
464   \begingroup
465   \def\bbl@tempe{l@}%
466   \edef\bbl@tempd{\noexpand\@ifundefined{\noexpand\bbl@tempe#1}}%
467   \bbl@tempd
468   {\lowercase\expandafter{\bbl@tempd}%
469    \uppercase\expandafter{\bbl@tempd}%
470    \@empty
471    {\edef\bbl@tempd{\def\noexpand#1{#1}}%
472     \uppercase\expandafter{\bbl@tempd}}}%
473    {\edef\bbl@tempd{\def\noexpand#1{#1}}%
474     \lowercase\expandafter{\bbl@tempd}}}%
475   \@empty
476   \edef\bbl@tempd{\endgroup\def\noexpand#1{#1}}%
477   \bbl@tempd
478   \bbl@exp{\bbl@usehooks{language}{\language}{#1}}}%
479 \def\bbl@iflanguage#1{%
480   \@ifundefined{l@#1}{\@nolanerr{#1}\@gobble}\@firstofone}

```

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

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

```

481 \def\bbl@bcpcase#1#2#3#4\@#5{%
482   \ifx\@empty#3%
483     \uppercase{\def#5{#1#2}}%
484   \else
485     \uppercase{\def#5{#1}}%
486     \lowercase{\edef#5{#5#2#3#4}}%
487   \fi}
488 \def\bbl@bcpllookup#1-#2-#3-#4\@{%
489   \let\bbl@bcp\relax
490   \lowercase{\def\bbl@tempa{#1}}%
491   \ifx\@empty#2%
492     \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
493   \else\ifx\@empty#3%
494     \bbl@bcpcase#2\@empty\@empty\@{\bbl@tempb
495     \IfFileExists{babel-\bbl@tempa-\bbl@tempb.ini}%
496     {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb}}%
497     }%
498     \ifx\bbl@bcp\relax
499       \IfFileExists{babel-\bbl@tempa.ini}{\let\bbl@bcp\bbl@tempa}{}%
500     \fi
501   \else
502     \bbl@bcpcase#2\@empty\@empty\@{\bbl@tempb
503     \bbl@bcpcase#3\@empty\@empty\@{\bbl@tempc
504     \IfFileExists{babel-\bbl@tempa-\bbl@tempb-\bbl@tempc.ini}%
505     {\edef\bbl@bcp{\bbl@tempa-\bbl@tempb-\bbl@tempc}}%
506     }%
507     \ifx\bbl@bcp\relax
508       \IfFileExists{babel-\bbl@tempa-\bbl@tempc.ini}%
509       {\edef\bbl@bcp{\bbl@tempa-\bbl@tempc}}%
510     }%

```

```

511 \fi
512 \ifx\bb1@bcp\relax
513 \IfFileExists{babel-\bb1@tempa-\bb1@tempc.ini}%
514 {\edef\bb1@bcp{\bb1@tempa-\bb1@tempc}}%
515 {}%
516 \fi
517 \ifx\bb1@bcp\relax
518 \IfFileExists{babel-\bb1@tempa.ini}{\let\bb1@bcp\bb1@tempa}{}%
519 \fi
520 \fi\fi}
521 \let\bb1@initoload\relax
522 \def\bb1@provide@locale{%
523 \ifx\babelprovide\@undefined
524 \bb1@error{For a language to be defined on the fly 'base'\\%
525 is not enough, and the whole package must be\\%
526 loaded. Either delete the 'base' option or\\%
527 request the languages explicitly}%
528 {See the manual for further details.}%
529 \fi
530 % TODO. Option to search if loaded, with \LocaleForEach
531 \let\bb1@auxname\language % Still necessary. TODO
532 \bb1@ifunset{\bb1@bcp@map@\language}{}% Move uplevel??
533 {\edef\language{\@nameuse{\bb1@bcp@map@\language}}}%
534 \ifbb1@bcpallowed
535 \expandafter\ifx\csname date\language\endcsname\relax
536 \expandafter
537 \bb1@bcplookup\language-\@empty-\@empty-\@empty\@@
538 \ifx\bb1@bcp\relax\else % Returned by \bb1@bcplookup
539 \edef\language{\bb1@bcp@prefix\bb1@bcp}%
540 \edef\localename{\bb1@bcp@prefix\bb1@bcp}%
541 \expandafter\ifx\csname date\language\endcsname\relax
542 \let\bb1@initoload\bb1@bcp
543 \bb1@exp{\@babelprovide[\bb1@autoload@bcptoptions]{\language}}%
544 \let\bb1@initoload\relax
545 \fi
546 \bb1@csarg\xdef{bcp@map@\bb1@bcp}{\localename}%
547 \fi
548 \fi
549 \fi
550 \expandafter\ifx\csname date\language\endcsname\relax
551 \IfFileExists{babel-\language.tex}%
552 {\bb1@exp{\@babelprovide[\bb1@autoload@options]{\language}}}%
553 {}%
554 \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.

```

555 \def\iflanguage#1{%
556 \bb1@iflanguage{#1}%
557 \ifnum\csname l@#1\endcsname=\language
558 \expandafter\@firstoftwo
559 \else
560 \expandafter\@secondoftwo
561 \fi}}

```

## 8.1 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.

```
562 \let\bbl@select@type\z@
563 \edef\selectlanguage{%
564   \noexpand\protect
565   \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`.

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

The following definition is preserved for backwards compatibility (eg, arabi, koma). It is related to a trick for 2.09, now discarded.

```
567 \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.

```
568 \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`

```
569 \def\bbl@push@language{%
570   \ifx\language\@undefined\else
571     \ifx\currentgrouplevel\@undefined
572       \xdef\bbl@language@stack{\language+\bbl@language@stack}%
573     \else
574       \ifnum\currentgrouplevel=\z@
575         \xdef\bbl@language@stack{\language+}%
576       \else
577         \xdef\bbl@language@stack{\language+\bbl@language@stack}%
578       \fi
579     \fi
580 \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`.

```
581 \def\bbl@pop@lang#1+#2\@{%
582   \edef\language{#1}%
583   \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).

```
584 \let\bbl@ifrestoring\@secondoftwo
585 \def\bbl@pop@language{%
586   \expandafter\bbl@pop@lang\bbl@language@stack\@@
587   \let\bbl@ifrestoring\@firstoftwo
588   \expandafter\bbl@set@language\expandafter{\language}%
589   \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).

```
590 \chardef\localeid\z@
591 \def\bbl@id@last{0} % No real need for a new counter
592 \def\bbl@id@assign{%
593   \bbl@ifunset{bbl@id@\language}%
594   {\count@\bbl@id@last\relax
595     \advance\count@\@ne
596     \bbl@csarg\chardef{id@\language}\count@
597     \edef\bbl@id@last{\the\count@}%
598     \ifcase\bbl@engine\or
599       \directlua{
600         Babel = Babel or {}
601         Babel.locale_props = Babel.locale_props or {}
602         Babel.locale_props[\bbl@id@last] = {}
603         Babel.locale_props[\bbl@id@last].name = '\language'
604       }%
605     \fi}%
606   }%
607   \chardef\localeid\bbl@ccl{id@}}
```

The unprotected part of `\selectlanguage`.

```
608 \expandafter\def\csname selectlanguage \endcsname#1{%
609   \ifnum\bbl@hymapset=\@cclv\let\bbl@hymapset\tw@\fi
610   \bbl@push@language
611   \aftergroup\bbl@pop@language
612   \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.

`\bbl@savelastskip` is used to deal with skips before the write whatsit (as suggested by U Fischer). Adapted from `hyperref`, but it might fail, so I'll consider it a temporary hack, while I study other options (the ideal, but very likely unfeasible except perhaps in `luatex`, is to avoid the `\write` altogether when not needed).

```
613 \def\BabelContentsFiles{toc,lof,lot}
614 \def\bbl@set@language#1{% from selectlanguage, pop@
```

```

615 % The old buggy way. Preserved for compatibility.
616 \edef\language{%
617   \ifnum\escapechar=\expandafter`\string#1\@empty
618   \else\string#1\@empty\fi}%
619 \ifcat\relax\noexpand#1%
620   \expandafter\ifx\csname date\language\endcsname\relax
621     \edef\language{#1}%
622     \let\locale\language
623   \else
624     \bbl@info{Using '\string\language' instead of 'language' is\\%
625               deprecated. If what you want is to use a\\%
626               macro containing the actual locale, make\\%
627               sure it does not not match any language.\\%
628               Reported}%
629     \ifx\scantokens\@undefined
630       \def\locale{??}%
631     \else
632       \scantokens\expandafter{\expandafter
633         \def\expandafter\locale\expandafter{\language}}%
634     \fi
635   \fi
636 \else
637   \def\locale{#1}% This one has the correct catcodes
638 \fi
639 \select@language{\language}%
640 % write to aux
641 \expandafter\ifx\csname date\language\endcsname\relax\else
642   \if@files
643     \ifx\babel@aux\@gobbletwo\else % Set if single in the first, redundant
644       \bbl@savelastskip
645       \protected@write\@auxout{}\string\babel@aux{\bbl@auxname}{}}%
646       \bbl@restorelastskip
647     \fi
648     \bbl@usehooks{write}{}%
649   \fi
650 \fi}
651 %
652 \let\bbl@restorelastskip\relax
653 \def\bbl@savelastskip{%
654   \let\bbl@restorelastskip\relax
655   \ifvmode
656     \ifdim\lastskip=\z@
657       \let\bbl@restorelastskip\nobreak
658     \else
659       \bbl@exp{%
660         \def\\bbl@restorelastskip{%
661           \skip@=\the\lastskip
662           \\nobreak \vskip-\skip@ \vskip\skip@}}%
663       \fi
664     \fi}
665 %
666 \newif\ifbbl@bcpallowed
667 \bbl@bcpallowedfalse
668 \def\select@language#1{% from set@, babel@aux
669 % set hmap
670 \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
671 % set name
672 \edef\language{#1}%
673 \bbl@fixname\language

```

```

674 % TODO. name@map must be here?
675 \bbl@provide@locale
676 \bbl@iflanguage\language\language\endcsname\relax
677 \expandafter\ifx\csname date\language\endcsname\relax
678 \bbl@error
679 {Unknown language '\language'. Either you have\\%
680 misspelled its name, it has not been installed,\\%
681 or you requested it in a previous run. Fix its name,\\%
682 install it or just rerun the file, respectively. In\\%
683 some cases, you may need to remove the aux file}%
684 {You may proceed, but expect wrong results}%
685 \else
686 % set type
687 \let\bbl@select@type\z@
688 \expandafter\bbl@switch\expandafter{\language}%
689 \fi}}
690 \def\babel@aux#1#2{%
691 \select@language{#1}%
692 \bbl@foreach\BabelContentsFiles{% \relax -> don't assume vertical mode
693 \@writefile{##1}{\babel@toc{#1}{#2}\relax}}}% TODO - plain?
694 \def\babel@toc#1#2{%
695 \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.

```

696 \newif\ifbbl@usedatagroup
697 \def\bbl@switch#1{% from select@, foreign@
698 % make sure there is info for the language if so requested
699 \bbl@ensureinfo{#1}%
700 % restore
701 \originalTeX
702 \expandafter\def\expandafter\originalTeX\expandafter{%
703 \csname noextras#1\endcsname
704 \let\originalTeX\@empty
705 \babel@beginsave}%
706 \bbl@usehooks{afterreset}}}%
707 \languageshorthands{none}%
708 % set the locale id
709 \bbl@id@assign
710 % switch captions, date
711 % No text is supposed to be added here, so we remove any
712 % spurious spaces.
713 \bbl@bsphack
714 \ifcase\bbl@select@type
715 \csname captions#1\endcsname\relax
716 \csname date#1\endcsname\relax
717 \else
718 \bbl@xin@{,captions,},{, \bbl@select@opts,}%
719 \ifin@

```



```

720     \csname captions#1\endcsname\relax
721     \fi
722     \bbl@xin@{,date,}{, \bbl@select@opts,}%
723     \ifin@ % if \foreign... within \<lang>date
724     \csname date#1\endcsname\relax
725     \fi
726     \fi
727     \bbl@esphack
728     % switch extras
729     \bbl@usehooks{beforeextras}{}%
730     \csname extras#1\endcsname\relax
731     \bbl@usehooks{afterextras}{}%
732     % > babel-ensure
733     % > babel-sh-<short>
734     % > babel-bidi
735     % > babel-fontspec
736     % hyphenation - case mapping
737     \ifcase\bbl@opt@hyphenmap\or
738     \def\BabelLower##1##2{\lccode##1=##2\relax}%
739     \ifnum\bbl@hymapsel>4\else
740     \csname\language\name @bbl@hyphenmap\endcsname
741     \fi
742     \chardef\bbl@opt@hyphenmap\z@
743     \else
744     \ifnum\bbl@hymapsel>\bbl@opt@hyphenmap\else
745     \csname\language\name @bbl@hyphenmap\endcsname
746     \fi
747     \fi
748     \let\bbl@hymapsel\@cclv
749     % hyphenation - select rules
750     \ifnum\csname l@\language\name\endcsname=\l@unhyphenated
751     \edef\bbl@tempa{u}%
752     \else
753     \edef\bbl@tempa{\bbl@c1{lnbrk}}%
754     \fi
755     % linebreaking - handle u, e, k (v in the future)
756     \bbl@xin@{/u}{/\bbl@tempa}%
757     \ifin@ \else \bbl@xin@{/e}{/\bbl@tempa} \fi % elongated forms
758     \ifin@ \else \bbl@xin@{/k}{/\bbl@tempa} \fi % only kashida
759     \ifin@ \else \bbl@xin@{/v}{/\bbl@tempa} \fi % variable font
760     \ifin@
761     % unhyphenated/kashida/elongated = allow stretching
762     \language\l@unhyphenated
763     \babel@savevariable\emergencystretch
764     \emergencystretch\maxdimen
765     \babel@savevariable\hbadness
766     \hbadness\@M
767     \else
768     % other = select patterns
769     \bbl@patterns{#1}%
770     \fi
771     % hyphenation - mins
772     \babel@savevariable\lefthyphenmin
773     \babel@savevariable\righthyphenmin
774     \expandafter\ifx\csname #1hyphenmins\endcsname\relax
775     \set@hyphenmins\tw@\thr@\relax
776     \else
777     \expandafter\expandafter\expandafter\set@hyphenmins
778     \csname #1hyphenmins\endcsname\relax

```

779 \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.

```
780 \long\def\otherlanguage#1{%
781 \ifnum\bbl@hymapsel=\@cclv\let\bbl@hymapsel\thr@@\fi
782 \csname selectlanguage \endcsname{#1}%
783 \ignorespaces}
```

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

```
784 \long\def\endotherlanguage{%
785 \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`.

```
786 \expandafter\def\csname otherlanguage*\endcsname{%
787 \@ifnextchar[\bbl@otherlanguage@s{\bbl@otherlanguage@s[]}}
788 \def\bbl@otherlanguage@s[#1]#2{%
789 \ifnum\bbl@hymapsel=\@cclv\chardef\bbl@hymapsel4\relax\fi
790 \def\bbl@select@opts{#1}%
791 \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”.

```
792 \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`.

```
793 \providecommand\bbl@beforeforeign{}
794 \edef\foreignlanguage{%
795 \noexpand\protect
796 \expandafter\noexpand\csname foreignlanguage \endcsname}
797 \expandafter\def\csname foreignlanguage \endcsname{%
```

```

798 \ifstar\bbbl@foreign@s\bbbl@foreign@x}
799 \providecommand\bbbl@foreign@x[3][[]{%
800 \begingroup
801 \def\bbbl@select@opts{#1}%
802 \let\BabelText\@firstofone
803 \bbbl@beforeforeign
804 \foreign@language{#2}%
805 \bbbl@usehooks{foreign}{}%
806 \BabelText{#3}% Now in horizontal mode!
807 \endgroup}
808 \def\bbbl@foreign@s#1#2{% TODO - \shapemode, \setpar, ?\@par
809 \begingroup
810 {\par}%
811 \let\bbbl@select@opts\@empty
812 \let\BabelText\@firstofone
813 \foreign@language{#1}%
814 \bbbl@usehooks{foreign*}{}%
815 \bbbl@dirparastext
816 \BabelText{#2}% Still in vertical mode!
817 {\par}%
818 \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 `bbbl@switch`.

```

819 \def\foreign@language#1{%
820 % set name
821 \edef\language#1}%
822 \ifbbbl@usedategroup
823 \bbbl@add\bbbl@select@opts{,date,}%
824 \bbbl@usedategroupfalse
825 \fi
826 \bbbl@fixname\language
827 % TODO. name@map here?
828 \bbbl@provide@locale
829 \bbbl@iflanguage\language{\language\relax
830 \expandafter\ifx\csname date\language\endcsname\relax
831 \bbbl@warning % TODO - why a warning, not an error?
832 {Unknown language '#1'. Either you have\\%
833 misspelled its name, it has not been installed,\\%
834 or you requested it in a previous run. Fix its name,\\%
835 install it or just rerun the file, respectively. In\\%
836 some cases, you may need to remove the aux file.\\%
837 I'll proceed, but expect wrong results.\\%
838 Reported}%
839 \fi
840 % set type
841 \let\bbbl@select@type\@ne
842 \expandafter\bbbl@switch\expandafter{\language}}

```

`\bbbl@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). `\bbbl@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.

```

843 \let\bbl@hyphlist\@empty
844 \let\bbl@hyphenation@\relax
845 \let\bbl@pttnlist\@empty
846 \let\bbl@patterns@\relax
847 \let\bbl@hymapsel=\ccclv
848 \def\bbl@patterns#1{%
849   \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
850     \csname l@#1\endcsname
851     \edef\bbl@tempa{#1}%
852   \else
853     \csname l@#1:\f@encoding\endcsname
854     \edef\bbl@tempa{#1:\f@encoding}%
855   \fi
856   \@expandtwoargs\bbl@usehooks{patterns}{{#1}{\bbl@tempa}}%
857   % > luatex
858   \@ifundefined{bbl@hyphenation@}{}{% Can be \relax!
859   \begingroup
860     \bbl@xin@{,\number\language,}{,\bbl@hyphlist}%
861     \ifin@else
862       \@expandtwoargs\bbl@usehooks{hyphenation}{{#1}{\bbl@tempa}}%
863       \hyphenation{%
864         \bbl@hyphenation@
865         \@ifundefined{bbl@hyphenation@#1}%
866         \@empty
867         {\space\csname bbl@hyphenation@#1\endcsname}}%
868       \xdef\bbl@hyphlist{\bbl@hyphlist\number\language,}%
869     \fi
870   \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*`.

```

871 \def\hyphenrules#1{%
872   \edef\bbl@tempf{#1}%
873   \bbl@fixname\bbl@tempf
874   \bbl@iflanguage\bbl@tempf{%
875     \expandafter\bbl@patterns\expandafter{\bbl@tempf}%
876     \ifx\languageshortands\undefined\else
877       \languageshortands{none}%
878     \fi
879     \expandafter\ifx\csname\bbl@tempf hyphenmins\endcsname\relax
880       \set@hyphenmins\tw@\thr@\relax
881     \else
882       \expandafter\expandafter\expandafter\set@hyphenmins
883       \csname\bbl@tempf hyphenmins\endcsname\relax
884     \fi}}
885 \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.

```

886 \def\providehyphenmins#1#2{%
887   \expandafter\ifx\csname #1hyphenmins\endcsname\relax
888     \@namedef{#1hyphenmins}{#2}%
889   \fi}

```

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

```

890 \def\set@hyphenmins#1#2{%
891   \lefthyphenmin#1\relax
892   \righthyphenmin#2\relax}

\ProvidesLanguage The identification code for each file is something that was introduced in LATEX 2ε. When the
command \ProvidesFile does not exist, a dummy definition is provided temporarily. For use in the
language definition file the command \ProvidesLanguage is defined by babel.
Depending on the format, ie, on if the former is defined, we use a similar definition or not.

893 \ifx\ProvidesFile\@undefined
894   \def\ProvidesLanguage#1[#2 #3 #4]{%
895     \wlog{Language: #1 #4 #3 <#2>}%
896   }
897 \else
898   \def\ProvidesLanguage#1{%
899     \begingroup
900     \catcode`\ 10 %
901     \@makeother\/%
902     \ifnextchar[%]
903       {\@provideslanguage{#1}}{\@provideslanguage{#1}[]}
904   \def\@provideslanguage#1[#2]{%
905     \wlog{Language: #1 #2}%
906     \expandafter\xdef\csname ver@#1.ldf\endcsname{#2}%
907   \endgroup}
908 \fi

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

909 \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.

910 \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’:

911 \providecommand\setlocale{%
912   \bbl@error
913   {Not yet available}%
914   {Find an armchair, sit down and wait}}
915 \let\uselocale\setlocale
916 \let\locale\setlocale
917 \let\selectlocale\setlocale
918 \let\localename\setlocale
919 \let\textlocale\setlocale
920 \let\textlanguage\setlocale
921 \let\languagetext\setlocale

```

## 8.2 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.

**\@nopatterns**

**\@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 L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub>, 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.

```

922 \edef\bbl@nulllanguage{\string\language=0}
923 \def\bbl@nocaption{\protect\bbl@nocaption@i}
924 \def\bbl@nocaption@i#1#2{% 1: text to be printed 2: caption macro \langXname
925   \global\@namedef{#2}{\textbf{?#1?}}}%
926   \@nameuse{#2}%
927   \edef\bbl@tempa{#1}%
928   \bbl@sreplace\bbl@tempa{name}{}}%
929   \bbl@warning{% TODO.
930     \@backslashchar#1 not set for '\language'. Please,\\%
931     define it after the language has been loaded\\%
932     (typically in the preamble) with:\\%
933     \string\setlocalecaption{\language}{\bbl@tempa}{..}\\%
934     Reported}}
935 \def\bbl@tentative{\protect\bbl@tentative@i}
936 \def\bbl@tentative@i#1{%
937   \bbl@warning{%
938     Some functions for '#1' are tentative.\\%
939     They might not work as expected and their behavior\\%
940     could change in the future.\\%
941     Reported}}
942 \def\@nolanerr#1{%
943   \bbl@error
944   {You haven't defined the language '#1' yet.\\%
945     Perhaps you misspelled it or your installation\\%
946     is not complete}%
947   {Your command will be ignored, type <return> to proceed}}
948 \def\@nopatterns#1{%
949   \bbl@warning
950   {No hyphenation patterns were preloaded for\\%
951     the language '#1' into the format.\\%
952     Please, configure your TeX system to add them and\\%
953     rebuild the format. Now I will use the patterns\\%
954     preloaded for \bbl@nulllanguage\space instead}}
955 \let\bbl@usehooks\@gobbletwo
956 \ifx\bbl@onlyswitch\@empty\endinput\fi
957 % Here ended switch.def

```

Here ended the now discarded switch.def. Here also (currently) ends the base option.

```

958 \ifx\directlua\@undefined\else
959   \ifx\bbl@luapatterns\@undefined
960     \input luababel.def
961   \fi
962 \fi
963 <<Basic macros>>
964 \bbl@trace{Compatibility with language.def}
965 \ifx\bbl@languages\@undefined
966   \ifx\directlua\@undefined
967     \openin1 = language.def % TODO. Remove hardcoded number
968     \ifeof1
969       \closein1
970       \message{I couldn't find the file language.def}
971     \else
972       \closein1
973       \begingroup
974         \def\addlanguage#1#2#3#4#5{%
975           \expandafter\ifx\csname lang@#1\endcsname\relax\else
976             \global\expandafter\let\csname l@#1\endcsname\expandafter\endcsname

```

```

977         \csname lang@#1\endcsname
978     \fi}%
979     \def\uselanguage#1{%
980         \input language.def
981     \endgroup
982 \fi
983 \fi
984 \chardef\l@english\z@
985 \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.

```

986 \def\addto#1#2{%
987     \ifx#1\undefined
988         \def#1{#2}%
989     \else
990         \ifx#1\relax
991             \def#1{#2}%
992         \else
993             {\toks@\expandafter{#1#2}%
994             \xdef#1{\the\toks@}}%
995         \fi
996     \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?

```

997 \def\bbl@withactive#1#2{%
998     \begingroup
999     \lccode`~=#2\relax
1000     \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 T<sub>E</sub>X 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`.

```

1001 \def\bbl@redefine#1{%
1002     \edef\bbl@tempa{\bbl@stripslash#1}%
1003     \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1004     \expandafter\def\csname\bbl@tempa\endcsname}
1005 \@onlypreamble\bbl@redefine

```

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

```

1006 \def\bbl@redefine@long#1{%
1007     \edef\bbl@tempa{\bbl@stripslash#1}%
1008     \expandafter\let\csname org@\bbl@tempa\endcsname#1%
1009     \expandafter\long\expandafter\def\csname\bbl@tempa\endcsname}
1010 \@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_`.

```

1011 \def\bbl@redefineroobust#1{%
1012     \edef\bbl@tempa{\bbl@stripslash#1}%
1013     \bbl@ifunset{\bbl@tempa\space}%

```

```

1014 {\expandafter\let\csname org@bbl@tempa\endcsname#1%
1015 \bbl@exp{\def\#1{\protect\<bbl@tempa\space>}}}%
1016 {\bbl@exp{\let\<org@bbl@tempa>\<bbl@tempa\space>}}}%
1017 \@namedef{\bbl@tempa\space}}
1018 \@onlypreamble\bbl@redefineroast

```

### 8.3 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.

```

1019 \bbl@trace{Hooks}
1020 \newcommand\AddBabelHook[3][{}]{%
1021 \bbl@ifunset{bbl@hk@#2}{\EnableBabelHook{#2}}{}%
1022 \def\bbl@tempa##1,#3=##2,##3\@empty{\def\bbl@tempb{##2}}%
1023 \expandafter\bbl@tempa\bbl@evargs,#3=,\@empty
1024 \bbl@ifunset{bbl@ev@#2@#3@#1}%
1025 {\bbl@csarg\bbl@add{ev@#3@#1}{\bbl@elth{#2}}}%
1026 {\bbl@csarg\let{ev@#2@#3@#1}\relax}%
1027 \bbl@csarg\newcommand{ev@#2@#3@#1}[\bbl@tempb]}
1028 \newcommand\EnableBabelHook[1]{\bbl@csarg\let{hk@#1}\@firstofone}
1029 \newcommand\DisableBabelHook[1]{\bbl@csarg\let{hk@#1}\@gobble}
1030 \def\bbl@usehooks#1#2{%
1031 \ifx\UseHook\@undefined\else\UseHook{babel/#1}\fi
1032 \def\bbl@elth##1{%
1033 \bbl@cs{hk@##1}{\bbl@cs{ev@##1@#1@#2}}%
1034 \bbl@cs{ev@#1@}%
1035 \ifx\language\@undefined\else % Test required for Plain (?)
1036 \ifx\UseHook\@undefined\else\UseHook{babel/#1/\language}\fi
1037 \def\bbl@elth##1{%
1038 \bbl@cs{hk@##1}{\bbl@cl{ev@##1@#1@#2}}%
1039 \bbl@cl{ev@#1}%
1040 \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).

```

1041 \def\bbl@evargs{,% <- don't delete this comma
1042 everylanguage=1,loadkernel=1,loadpatterns=1,loadexceptions=1,%
1043 addialect=2,patterns=2,defaultcommands=0,encodedcommands=2,write=0,%
1044 beforeextras=0,afterextras=0,stopcommands=0,stringprocess=0,%
1045 hyphenation=2,initiateactive=3,afterreset=0,foreign=0,foreign*=0,%
1046 beforestart=0,language=2}
1047 \ifx\NewHook\@undefined\else
1048 \def\bbl@tempa#1=#2\@{\NewHook{babel/#1}}
1049 \bbl@foreach\bbl@evargs{\bbl@tempa#1\@}
1050 \fi

```

`\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.

```

1051 \bbl@trace{Defining babelensure}

```



```

1052 \newcommand\babelensure[2][\% TODO - revise test files
1053 \AddBabelHook{babel-ensure}{afterextras}{\%
1054 \ifcase\bbbl@select@type
1055 \bbbl@cl{e}%
1056 \fi}%
1057 \beginingroup
1058 \let\bbbl@ens@include\@empty
1059 \let\bbbl@ens@exclude\@empty
1060 \def\bbbl@ens@fontenc{\relax}%
1061 \def\bbbl@tempb##1{\%
1062 \ifx\@empty##1\else\noexpand##1\expandafter\bbbl@tempb\fi}%
1063 \edef\bbbl@tempa{\bbbl@tempb##1\@empty}%
1064 \def\bbbl@tempb##1=##2\@{\@namedef{\bbbl@ens@##1}{##2}}%
1065 \bbbl@foreach\bbbl@tempa{\bbbl@tempb##1\@}%
1066 \def\bbbl@tempc{\bbbl@ensure}%
1067 \expandafter\bbbl@add\expandafter\bbbl@tempc\expandafter{\%
1068 \expandafter{\bbbl@ens@include}}%
1069 \expandafter\bbbl@add\expandafter\bbbl@tempc\expandafter{\%
1070 \expandafter{\bbbl@ens@exclude}}%
1071 \toks@\expandafter{\bbbl@tempc}%
1072 \bbbl@exp{%
1073 \endgroup
1074 \def\<bbbl@e@#2>{\the\toks@{\bbbl@ens@fontenc}}}%
1075 \def\bbbl@ensure#1#2#3{\% 1: include 2: exclude 3: fontenc
1076 \def\bbbl@tempb##1{\% elt for (excluding) \bbbl@captionslist list
1077 \ifx##1\@undefined \% 3.32 - Don't assume the macro exists
1078 \edef##1{\noexpand\bbbl@nocaption
1079 {\bbbl@stripslash##1}{\language\bbbl@stripslash##1}}%
1080 \fi
1081 \ifx##1\@empty\else
1082 \in@{##1}{#2}%
1083 \ifin@ \else
1084 \bbbl@ifunset{\bbbl@ensure@\language\bbbl@stripslash##1}%
1085 {\bbbl@exp{%
1086 \\\DeclareRobustCommand\<bbbl@ensure@\language\bbbl@stripslash##1>[1]{\%
1087 \\\foreignlanguage{\language\bbbl@stripslash##1}%
1088 {\ifx\relax#3\else
1089 \\\fontencoding{#3}\selectfont
1090 \fi
1091 #####1}}}%
1092 }%
1093 \toks@\expandafter{##1}%
1094 \edef##1{\%
1095 \bbbl@csarg\noexpand{\bbbl@ensure@\language\bbbl@stripslash##1}%
1096 {\the\toks@}}%
1097 \fi
1098 \expandafter\bbbl@tempb
1099 \fi}%
1100 \expandafter\bbbl@tempb\bbbl@captionslist\today\@empty
1101 \def\bbbl@tempa##1{\% elt for include list
1102 \ifx##1\@empty\else
1103 \bbbl@csarg\in@{\bbbl@ensure@\language\bbbl@stripslash##1}\expandafter{\bbbl@tempa##1}%
1104 \ifin@ \else
1105 \bbbl@tempb##1\@empty
1106 \fi
1107 \expandafter\bbbl@tempa
1108 \fi}%
1109 \bbbl@tempa##1\@empty}
1110 \def\bbbl@captionslist{\%

```

```

1111 \prefacename\refname\abstractname\bibname\chaptername\appendixname
1112 \contentsname\listfigurename\listtablename\indexname\figurename
1113 \tablename\partname\enclname\ccname\headtoname\pagename\seename
1114 \alsoname\proofname\glossaryname}

```

## 8.4 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 `@`-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`.

```

1115 \bbl@trace{Macros for setting language files up}
1116 \def\bbl@ldfinit{%
1117   \let\bbl@screset@empty
1118   \let\BabelStrings\bbl@opt@string
1119   \let\BabelOptions@empty
1120   \let\BabelLanguages\relax
1121   \ifx\originalTeX\@undefined
1122     \let\originalTeX@empty
1123   \else
1124     \originalTeX
1125   \fi}
1126 \def\LdfInit#1#2{%
1127   \chardef\atcatcode=\catcode`@
1128   \catcode`\@=11\relax
1129   \chardef\eqcatcode=\catcode`=
1130   \catcode`\==12\relax
1131   \expandafter\if\expandafter\@backslashchar
1132     \expandafter\@car\string#2\@nil
1133     \ifx#2\@undefined\else
1134       \ldf@quit{#1}%
1135     \fi
1136   \else
1137     \expandafter\ifx\csname#2\endcsname\relax\else
1138       \ldf@quit{#1}%
1139     \fi
1140   \fi
1141   \bbl@ldfinit}

```

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

```

1142 \def\ldf@quit#1{%
1143   \expandafter\main@language\expandafter{#1}%
1144   \catcode`\@=\atcatcode \let\atcatcode\relax
1145   \catcode`\==\eqcatcode \let\eqcatcode\relax

```

```

1146 \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.

1147 \def\bbl@afterldf#1{% TODO. Merge into the next macro? Unused elsewhere
1148 \bbl@afterlang
1149 \let\bbl@afterlang\relax
1150 \let\BabelModifiers\relax
1151 \let\bbl@screset\relax}%
1152 \def\ldf@finish#1{%
1153 \loadlocalcfg{#1}%
1154 \bbl@afterldf{#1}%
1155 \expandafter\main@language\expandafter{#1}%
1156 \catcode`\@=\atcatcode \let\atcatcode\relax
1157 \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.

1158 \@onlypreamble\LdfInit
1159 \@onlypreamble\ldf@quit
1160 \@onlypreamble\ldf@finish

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

1161 \def\main@language#1{%
1162 \def\bbl@main@language{#1}%
1163 \let\language\bbl@main@language % TODO. Set localename
1164 \bbl@id@assign
1165 \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.

1166 \def\bbl@beforestart{%
1167 \def\@nolanerr##1{%
1168 \bbl@warning{Undefined language '##1' in aux.\Reported}}%
1169 \bbl@usehooks{beforestart}}}%
1170 \global\let\bbl@beforestart\relax}
1171 \AtBeginDocument{%
1172 {\@nameuse{bbl@beforestart}}% Group!
1173 \if@filesw
1174 \providecommand\babel@aux[2]{}%
1175 \immediate\write\@mainaux{%
1176 \string\providecommand\string\babel@aux[2]{}%
1177 \immediate\write\@mainaux{\string\@nameuse{bbl@beforestart}}%
1178 \fi
1179 \expandafter\selectlanguage\expandafter{\bbl@main@language}%
1180 \ifbbl@single % must go after the line above.
1181 \renewcommand\selectlanguage[1]{}%
1182 \renewcommand\foreignlanguage[2]{#2}%
1183 \global\let\babel@aux\@gobbletwo % Also as flag
1184 \fi
1185 \ifcase\bbl@engine\or\pagedir\bodydir\fi} % TODO - a better place

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

```

```

1186 \def\select@language#1{%
1187   \ifcase\bbbl@select@type
1188     \bbbl@ifsamestring\languagename{#1}{\select@language{#1}}%
1189   \else
1190     \select@language{#1}%
1191   \fi}

```

## 8.5 Shorthands

`\bbbl@add@special` The macro `\bbbl@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.

```

1192 \bbbl@trace{Shorhands}
1193 \def\bbbl@add@special#1{% 1:a macro like \", \?, etc.
1194   \bbbl@add\dospecials{\do#1}% test \@sanitize = \relax, for back. compat.
1195   \bbbl@ifunset{\@sanitize}{\bbbl@add\@sanitize{\@makeother#1}}%
1196   \ifx\nfss@catcodes\@undefined\else % TODO - same for above
1197     \begingroup
1198       \catcode`#1\active
1199       \nfss@catcodes
1200       \ifnum\catcode`#1=\active
1201         \endgroup
1202       \bbbl@add\nfss@catcodes{\@makeother#1}%
1203     \else
1204       \endgroup
1205     \fi
1206   \fi}

```

`\bbbl@remove@special` The companion of the former macro is `\bbbl@remove@special`. It removes a character from the set macros `\dospecials` and `\@sanitize`, but it is not used at all in the babel core.

```

1207 \def\bbbl@remove@special#1{%
1208   \begingroup
1209     \def\x##1##2{\ifnum`#1=``##2\noexpand\@empty
1210       \else\noexpand##1\noexpand##2\fi}%
1211     \def\do{\x\do}%
1212     \def\@makeother{\x\@makeother}%
1213   \edef\x{\endgroup
1214     \def\noexpand\dospecials{\dospecials}%
1215     \expandafter\ifx\csgname \@sanitize\endcsgname\relax\else
1216       \def\noexpand\@sanitize{\@sanitize}%
1217     \fi}%
1218   \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 `\bbbl@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).

```

1219 \def\bbl@active@def#1#2#3#4{%
1220   \namedef{#3#1}{%
1221     \expandafter\ifx\csname#2@sh@#1\endcsname\relax
1222       \bbl@afterelse\bbl@sh@select#2#1{#3@arg#1}{#4#1}%
1223     \else
1224       \bbl@afterfi\csname#2@sh@#1\endcsname
1225     \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.

```

1226   \long\@namedef{#3@arg#1}##1{%
1227     \expandafter\ifx\csname#2@sh@#1@string##1\endcsname\relax
1228       \bbl@afterelse\csname#4#1\endcsname##1%
1229     \else
1230       \bbl@afterfi\csname#2@sh@#1@string##1\endcsname
1231     \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.

```

1232 \def\initiate@active@char#1{%
1233   \bbl@ifunset{active@char\string#1}%
1234   {\bbl@withactive
1235     {\expandafter\@initiate@active@char\expandafter}#1\string#1}%
1236   {}}

```

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` and preserving some degree of protection).

```

1237 \def\@initiate@active@char#1#2#3{%
1238   \bbl@csarg\edef{oricat@#2}{\catcode`#2=\the\catcode`#2\relax}%
1239   \ifx#1\@undefined
1240     \bbl@csarg\def{oridef@#2}{\def#1{\active@prefix#1\@undefined}}%
1241   \else
1242     \bbl@csarg\let{oridef@#2}#1%
1243     \bbl@csarg\edef{oridef@#2}{%
1244       \let\noexpand#1%
1245       \expandafter\noexpand\csname bbl@oridef@@#2\endcsname}%
1246   \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*").

```

1247   \ifx#1#3\relax
1248     \expandafter\let\csname normal@char#2\endcsname#3%
1249   \else
1250     \bbl@info{Making #2 an active character}%
1251     \ifnum\mathcode`#2=\ifodd\bbl@engine"1000000 \else"8000 \fi
1252     \@namedef{normal@char#2}{%
1253       \textormath{#3}{\csname bbl@oridef@@#2\endcsname}}%
1254   \else

```

```

1255 \namedef{normal@char#2}{#3}%
1256 \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).

```

1257 \bbl@restoreactive{#2}%
1258 \AtBeginDocument{%
1259 \catcode`#2\active
1260 \if@filesw
1261 \immediate\write\@mainaux{\catcode`\string#2\active}%
1262 \fi}%
1263 \expandafter\bbl@add@special\csname#2\endcsname
1264 \catcode`#2\active
1265 \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⟩`).

```

1266 \let\bbl@tempa\@firstoftwo
1267 \if\string^#2%
1268 \def\bbl@tempa{\noexpand\textormath}%
1269 \else
1270 \ifx\bbl@mathnormal\@undefined\else
1271 \let\bbl@tempa\bbl@mathnormal
1272 \fi
1273 \fi
1274 \expandafter\edef\csname active@char#2\endcsname{%
1275 \bbl@tempa
1276 {\noexpand\if@safe@actives
1277 \noexpand\expandafter
1278 \expandafter\noexpand\csname normal@char#2\endcsname
1279 \noexpand\else
1280 \noexpand\expandafter
1281 \expandafter\noexpand\csname bbl@doactive#2\endcsname
1282 \noexpand\fi}%
1283 {\expandafter\noexpand\csname normal@char#2\endcsname}}}%
1284 \bbl@csarg\edef{doactive#2}{%
1285 \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

$$\backslash active@prefix \langle char \rangle \backslash normal@char \langle char \rangle$$

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

```

1286 \bbl@csarg\edef{active@#2}{%
1287 \noexpand\active@prefix\noexpand#1%
1288 \expandafter\noexpand\csname active@char#2\endcsname}%
1289 \bbl@csarg\edef{normal@#2}{%
1290 \noexpand\active@prefix\noexpand#1%
1291 \expandafter\noexpand\csname normal@char#2\endcsname}%
1292 \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.

```
1293 \bbl@active@def#2\user@group{user@active}{language@active}%
1294 \bbl@active@def#2\language@group{language@active}{system@active}%
1295 \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.

```
1296 \expandafter\edef\csname\user@group @sh#2@@\endcsname
1297 {\expandafter\noexpand\csname normal@char#2\endcsname}%
1298 \expandafter\edef\csname\user@group @sh#2@\string\protect\endcsname
1299 {\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.

```
1300 \if\string'#2%
1301 \let\prim@s\bbl@prim@s
1302 \let\active@math@prime#1%
1303 \fi
1304 \bbl@usehooks{initiateactive}{\{#1\}{#2\}{#3\}}
```

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

```
1305 <<(*More package options)>> ≡
1306 \DeclareOption{math=active}{}
1307 \DeclareOption{math=normal}{\def\bbl@mathnormal{\noexpand\textormath}}
1308 <</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.

```
1309 \@ifpackagewith{babel}{KeepShorthandsActive}%
1310 {\let\bbl@restoreactive\@gobble}%
1311 {\def\bbl@restoreactive#1{%
1312 \bbl@exp{%
1313 \\\AfterBabelLanguage\\CurrentOption
1314 {\catcode`#1=\the\catcode`#1\relax}%
1315 \\\AtEndOfPackage
1316 {\catcode`#1=\the\catcode`#1\relax}}}%
1317 \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.

```
1318 \def\bbl@sh@select#1#2{%
1319 \expandafter\ifx\csname#1@sh#2@sel\endcsname\relax
1320 \bbl@afterelse\bbl@scndcs
1321 \else
1322 \bbl@afterfi\csname#1@sh#2@sel\endcsname
1323 \fi}
```

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

```

1324 \begingroup
1325 \bbl@ifunset{ifincsname}% TODO. Ugly. Correct? Only Plain?
1326 {\gdef\active@prefix#1{%
1327   \ifx\protect\@typeset@protect
1328   \else
1329     \ifx\protect\@unexpandable@protect
1330     \noexpand#1%
1331     \else
1332     \protect#1%
1333     \fi
1334     \expandafter\@gobble
1335     \fi}}
1336 {\gdef\active@prefix#1{%
1337   \ifincsname
1338   \string#1%
1339   \expandafter\@gobble
1340   \else
1341     \ifx\protect\@typeset@protect
1342     \else
1343       \ifx\protect\@unexpandable@protect
1344       \noexpand#1%
1345       \else
1346       \protect#1%
1347       \fi
1348       \expandafter\expandafter\expandafter\@gobble
1349       \fi
1350     \fi}}
1351 \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>`.

```

1352 \newif\if@safe@actives
1353 \@safe@activesfalse

```

`\bbl@restore@actives` When the output routine kicks in while the active characters were made “safe” this must be undone in the headers to prevent unexpected typeset results. For this situation we define a command to make them “unsafe” again.

```

1354 \def\bbl@restore@actives{\if@safe@actives\@safe@activesfalse\fi}

```

`\bbl@activate` Both macros take one argument, like `\initiate@active@char`. The macro is used to change the definition of an active character to expand to `\active@char<char>` in the case of `\bbl@activate`, or `\normal@char<char>` in the case of `\bbl@deactivate`.

```

1355 \chardef\bbl@activated\z@
1356 \def\bbl@activate#1{%
1357   \chardef\bbl@activated\ne
1358   \bbl@withactive{\expandafter\let\expandafter}#1%
1359   \csname bbl@active@\string#1\endcsname}
1360 \def\bbl@deactivate#1{%
1361   \chardef\bbl@activated\tw@
1362   \bbl@withactive{\expandafter\let\expandafter}#1%
1363   \csname bbl@normal@\string#1\endcsname}

```



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

`\bbl@scndcs`

```

1364 \def\bbl@firstcs#1#2{\csname#1\endcsname}
1365 \def\bbl@scndcs#1#2{\csname#2\endcsname}

```

`\declare@shorthand` The command `\declare@shorthand` is used to declare a shorthand on a certain level. It takes three arguments:

1. a name for the collection of shorthands, i.e. ‘system’, or ‘dutch’;
2. the character (sequence) that makes up the shorthand, i.e. ~ or "a;
3. the code to be executed when the shorthand is encountered.

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 `ldf` files.

```

1366 \def\babel@texpdf#1#2#3#4{%
1367   \ifx\texorpdfstring\undefined
1368     \textormath{#1}{#3}%
1369   \else
1370     \texorpdfstring{\textormath{#1}{#3}}{#2}%
1371     % \texorpdfstring{\textormath{#1}{#3}}{\textormath{#2}{#4}}%
1372   \fi}
1373 %
1374 \def\declare@shorthand#1#2{\@decl@short{#1}#2\@nil}
1375 \def\@decl@short#1#2#3\@nil#4{%
1376   \def\bbl@tempa{#3}%
1377   \ifx\bbl@tempa\@empty
1378     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@scndcs
1379     \bbl@ifunset{#1@sh@\string#2@}{}%
1380     {\def\bbl@tempa{#4}%
1381      \expandafter\ifx\csname#1@sh@\string#2@\endcsname\bbl@tempa
1382      \else
1383        \bbl@info
1384        {Redefining #1 shorthand \string#2\%
1385         in language \CurrentOption}%
1386      \fi}%
1387     \@namedef{#1@sh@\string#2@}{#4}%
1388   \else
1389     \expandafter\let\csname #1@sh@\string#2@sel\endcsname\bbl@firstcs
1390     \bbl@ifunset{#1@sh@\string#2@\string#3@}{}%
1391     {\def\bbl@tempa{#4}%
1392      \expandafter\ifx\csname#1@sh@\string#2@\string#3@\endcsname\bbl@tempa
1393      \else
1394        \bbl@info
1395        {Redefining #1 shorthand \string#2\string#3\%
1396         in language \CurrentOption}%
1397      \fi}%
1398     \@namedef{#1@sh@\string#2@\string#3@}{#4}%
1399   \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.

```

1400 \def\textormath{%
1401   \ifmmode
1402     \expandafter\@secondoftwo
1403   \else
1404     \expandafter\@firstoftwo
1405   \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’.

```

1406 \def\user@group{user}
1407 \def\language@group{english} % TODO. I don't like defaults
1408 \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.

```

1409 \def\useshorthands{%
1410   \@ifstar\bbbl@usesh@s{\bbbl@usesh@x{}}
1411 \def\bbbl@usesh@s#1{%
1412   \bbbl@usesh@x
1413   {\AddBabelHook{babel-sh-\string#1}{afterextras}{\bbbl@activate{#1}}}%
1414   {#1}}
1415 \def\bbbl@usesh@x#1#2{%
1416   \bbbl@ifshorthand{#2}%
1417   {\def\user@group{user}%
1418     \initiate@active@char{#2}%
1419     #1%
1420     \bbbl@activate{#2}}%
1421   {\bbbl@error
1422     {I can't declare a shorthand turned off (\string#2)}
1423     {Sorry, but you can't use shorthands which have been\\
1424       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 `\bbbl@set@user@generic`); we make also sure `{}` and `\protect` are taken into account in this new top level.

```

1425 \def\user@language@group{user@\language@group}
1426 \def\bbbl@set@user@generic#1#2{%
1427   \bbbl@ifunset{user@generic@active#1}%
1428   {\bbbl@active@def#1\user@language@group{user@active}{user@generic@active}%
1429     \bbbl@active@def#1\user@group{user@generic@active}{language@active}%
1430     \expandafter\edef\csname#2@sh@#1@@\endcsname{%
1431       \expandafter\noexpand\csname normal@char#1\endcsname}%
1432     \expandafter\edef\csname#2@sh@#1@\string\protect@\endcsname{%
1433       \expandafter\noexpand\csname user@active#1\endcsname}}%
1434   \@empty}
1435 \newcommand\defineshorthand[3][user]{%
1436   \edef\bbbl@tempa{\zap@space#1 \@empty}%
1437   \bbbl@for\bbbl@tempb\bbbl@tempa{%
1438     \if*\expandafter\@car\bbbl@tempb\@nil
1439       \edef\bbbl@tempb{user@\expandafter\@gobble\bbbl@tempb}%
1440       \@expandtwoargs
1441       \bbbl@set@user@generic{\expandafter\string\@car#2\@nil}\bbbl@tempb
1442     \fi
1443     \declare@shorthand{\bbbl@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].

```

1444 \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`.

```

1445 \def\aliasshorthand#1#2{%
1446   \bbl@ifshorthand{#2}%
1447   {\expandafter\ifx\csname active@char\string#2\endcsname\relax
1448     \ifx\document\@notprerr
1449       \@notshorthand{#2}%
1450     \else
1451       \initiate@active@char{#2}%
1452       \expandafter\let\csname active@char\string#2\expandafter\endcsname
1453         \csname active@char\string#1\endcsname
1454       \expandafter\let\csname normal@char\string#2\expandafter\endcsname
1455         \csname normal@char\string#1\endcsname
1456       \bbl@activate{#2}%
1457     \fi
1458   \fi}%
1459 {\bbl@error
1460   {Cannot declare a shorthand turned off (\string#2)}
1461   {Sorry, but you cannot use shorthands which have been\\
1462     turned off in the package options}}}

```

`\@notshorthand`

```

1463 \def\@notshorthand#1{%
1464   \bbl@error{%
1465     The character '\string #1' should be made a shorthand character;\\
1466     add the command \string\usesshorthands\string{#1\string} to
1467     the preamble.\\
1468     I will ignore your instruction}%
1469   {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.

```

1470 \newcommand*\shorthandon[1]{\bbl@switch@sh\@ne#1\@nnil}
1471 \DeclareRobustCommand*\shorthandoff{%
1472   \@ifstar{\bbl@shorthandoff\tw@}{\bbl@shorthandoff\z@}}
1473 \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.

```

1474 \def\bbl@switch@sh#1#2{%
1475   \ifx#2\@nnil\else
1476     \bbl@ifunset{\bbl@active@\string#2}%
1477     {\bbl@error
1478       {I can't switch '\string#2' on or off--not a shorthand}%
1479       {This character is not a shorthand. Maybe you made\\
1480         a typing mistake? I will ignore your instruction.}}}%
1481     {\ifcase#1%   off, on, off*
1482       \catcode`#2\relax
1483     \or
1484       \catcode`#2\active
1485       \bbl@ifunset{\bbl@shdef@\string#2}%
1486       {}}%

```

```

1487         {\bbl@withactive{\expandafter\let\expandafter}\#2%
1488         \csname bbl@shdef@\string#2\endcsname
1489         \bbl@csarg\let{shdef@\string#2}\relax}%
1490         \ifcase\bbl@activated\or
1491         \bbl@activate{#2}%
1492         \else
1493         \bbl@deactivate{#2}%
1494         \fi
1495     \or
1496     \bbl@ifunset{\bbl@shdef@\string#2}%
1497     {\bbl@withactive{\bbl@csarg\let{shdef@\string#2}}\#2}%
1498     {}%
1499     \csname bbl@oricat@\string#2\endcsname
1500     \csname bbl@oridef@\string#2\endcsname
1501     \fi}%
1502     \bbl@afterfi\bbl@switch@sh#1%
1503 \fi}

```

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

```

1504 \def\babelshorthand{\active@prefix\babelshorthand\bbl@putsh}
1505 \def\bbl@putsh#1{%
1506     \bbl@ifunset{\bbl@active@\string#1}%
1507     {\bbl@putsh@i#1\@empty\@nnil}%
1508     {\csname bbl@active@\string#1\endcsname}}
1509 \def\bbl@putsh@i#1#2\@nnil{%
1510     \csname\language@group @sh@\string#1@%
1511     \ifx\@empty#2\else\string#2\fi\endcsname}
1512 \ifx\bbl@opt@shorthands\@nnil\else
1513     \let\bbl@s@initiate@active@char\initiate@active@char
1514     \def\initiate@active@char#1{%
1515         \bbl@ifshorthand{#1}{\bbl@s@initiate@active@char{#1}}{}}
1516     \let\bbl@s@switch@sh\bbl@switch@sh
1517     \def\bbl@switch@sh#1#2{%
1518         \ifx#2\@nnil\else
1519         \bbl@afterfi
1520         \bbl@ifshorthand{#2}{\bbl@s@switch@sh#1{#2}}{\bbl@switch@sh#1}%
1521         \fi}
1522     \let\bbl@s@activate\bbl@activate
1523     \def\bbl@activate#1{%
1524         \bbl@ifshorthand{#1}{\bbl@s@activate{#1}}{}}
1525     \let\bbl@s@deactivate\bbl@deactivate
1526     \def\bbl@deactivate#1{%
1527         \bbl@ifshorthand{#1}{\bbl@s@deactivate{#1}}{}}
1528 \fi

```

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

```

1529 \newcommand\ifbabelshorthand[3]{\bbl@ifunset{\bbl@active@\string#1}{#3}{#2}}

```

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

```

1530 \def\bbl@prim@s{%
1531     \prime\futurelet\@let@token\bbl@pr@m@s}
1532 \def\bbl@if@primes#1#2{%
1533     \ifx#1\@let@token
1534     \expandafter\@firstoftwo
1535     \else\ifx#2\@let@token

```

```

1536 \bbl@afterelse\expandafter\@firstoftwo
1537 \else
1538 \bbl@afterfi\expandafter\@secondoftwo
1539 \fi\fi}
1540 \begingroup
1541 \catcode`\^=7 \catcode`\*= \active \lccode`\*=`\^
1542 \catcode`\'=12 \catcode`\`= \active \lccode`\`=``'
1543 \lowercase{%
1544 \gdef\bbl@pr@m@s{%
1545 \bbl@if@primes"%
1546 \pr@@s
1547 {\bbl@if@primes*^ \pr@@t\egroup}}
1548 \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).

```

1549 \initiate@active@char{~}
1550 \declare@shorthand{system}{~}{\leavevmode\nobreak\ }
1551 \bbl@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.

```

1552 \expandafter\def\csname OT1dqpos\endcsname{127}
1553 \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

```

1554 \ifx\f@encoding\@undefined
1555 \def\f@encoding{OT1}
1556 \fi

```

## 8.6 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.

```

1557 \bbl@trace{Language attributes}
1558 \newcommand\languageattribute[2]{%
1559 \def\bbl@tempc{#1}%
1560 \bbl@fixname\bbl@tempc
1561 \bbl@iflanguage\bbl@tempc{%
1562 \bbl@vforeach{#2}{%

```

We want to make sure that each attribute is selected only once; therefore we store the already selected attributes in `\bbl@known@attribs`. When that control sequence is not yet defined this attribute is certainly not selected before.

```

1563 \ifx\bbl@known@attribs\@undefined
1564 \in@false
1565 \else
1566 \bbl@xin@{,\bbl@tempc-##1,}{,\bbl@known@attribs,}%
1567 \fi

```

```

1568     \ifin@
1569     \bbl@warning{%
1570         You have more than once selected the attribute '##1'\%
1571         for language #1. Reported}%
1572     \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.

1573     \bbl@exp{%
1574         \\bbl@add@list\\bbl@known@attribs{\bbl@tempc-##1}}%
1575     \edef\bbl@tempa{\bbl@tempc-##1}%
1576     \expandafter\bbl@ifknown@ttrib\expandafter{\bbl@tempa}\bbl@attributes%
1577     {\csname\bbl@tempc @attr##1\endcsname}%
1578     {\@attrerr{\bbl@tempc}{##1}}%
1579     \fi}}
1580 \@onlypreamble\languageattribute

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

1581 \newcommand*{\@attrerr}[2]{%
1582     \bbl@error
1583     {The attribute #2 is unknown for language #1.}%
1584     {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}`.

```

1585 \def\bbl@declare@ttribute#1#2#3{%
1586     \bbl@xin@{,#2,}{,\BabelModifiers,}%
1587     \ifin@
1588         \AfterBabelLanguage{#1}{\languageattribute{#1}{#2}}%
1589     \fi
1590     \bbl@add@list\bbl@attributes{#1-#2}%
1591     \expandafter\def\csname#1@attr#2\endcsname{#3}}

```

`\bbl@ifattributeset` This internal macro has 4 arguments. It can be used to interpret T<sub>E</sub>X 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.

```

1592 \def\bbl@ifattributeset#1#2#3#4{%
1593     \ifx\bbl@known@attribs\undefined
1594         \in@false
1595     \else
1596         \bbl@xin@{,#1-#2,}{,\bbl@known@attribs,}%
1597     \fi
1598     \ifin@
1599         \bbl@afterelse#3%
1600     \else
1601         \bbl@afterfi#4%
1602     \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 T<sub>E</sub>X-code to be executed when the attribute is known and the T<sub>E</sub>X-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.

```

1603 \def\bbl@ifknown@ttrib#1#2{%

```

```

1604 \let\bbl@tempa\@secondoftwo
1605 \bbl@loopx\bbl@tempb{#2}{%
1606   \expandafter\in\expandafter{\expandafter,\bbl@tempb,}{, #1,}%
1607   \ifin@
1608     \let\bbl@tempa\@firstoftwo
1609   \else
1610     \fi}%
1611 \bbl@tempa}

```

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

```

1612 \def\bbl@clear@ttribs{%
1613   \ifx\bbl@attributes\undefined\else
1614     \bbl@loopx\bbl@tempa{\bbl@attributes}{%
1615       \expandafter\bbl@clear@ttrib\bbl@tempa.
1616     }%
1617     \let\bbl@attributes\undefined
1618   \fi}
1619 \def\bbl@clear@ttrib#1-#2.{%
1620   \expandafter\let\csname#1@attr@#2\endcsname\undefined}
1621 \AtBeginDocument{\bbl@clear@ttribs}

```

## 8.7 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`

```

1622 \bbl@trace{Macros for saving definitions}
1623 \def\babel@beginsave{\babel@savecnt\z@}

```

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

```

1624 \newcount\babel@savecnt
1625 \babel@beginsave

```

`\babel@save` The macro `\babel@save{csname}` saves the current meaning of the control sequence `csname` to `\originalTeX`<sup>31</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.

```

1626 \def\babel@save#1{%
1627   \expandafter\let\csname babel@number\babel@savecnt\endcsname#1\relax
1628   \toks@\expandafter{\originalTeX\let#1=}%
1629   \bbl@exp{%
1630     \def\\originalTeX{\the\toks@<\babel@number\babel@savecnt>\relax}}%
1631   \advance\babel@savecnt\@ne}
1632 \def\babel@savevariable#1{%
1633   \toks@\expandafter{\originalTeX #1=}%
1634   \bbl@exp{\def\\originalTeX{\the\toks@the#1\relax}}}

```

<sup>31</sup>`\originalTeX` has to be expandable, i.e. you shouldn't let it to `\relax`.

`\bbl@frenchspacing` Some languages need to have `\frenchspacing` in effect. Others don't want that. The command `\bbl@nonfrenchspacing` 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.

```

1635 \def\bbl@frenchspacing{%
1636   \ifnum\the\sfcode`\.=\@m
1637     \let\bbl@nonfrenchspacing\relax
1638   \else
1639     \frenchspacing
1640     \let\bbl@nonfrenchspacing\nonfrenchspacing
1641   \fi}
1642 \let\bbl@nonfrenchspacing\nonfrenchspacing
1643 \let\bbl@elt\relax
1644 \edef\bbl@fs@chars{%
1645   \bbl@elt{\string.}\@m{3000}\bbl@elt{\string?}\@m{3000}%
1646   \bbl@elt{\string!}\@m{3000}\bbl@elt{\string:}\@m{2000}%
1647   \bbl@elt{\string;}\@m{1500}\bbl@elt{\string,}\@m{1250}}
1648 \def\bbl@pre@fs{%
1649   \def\bbl@elt##1##2##3{\sfcode`##1=\the\sfcode`##1\relax}%
1650   \edef\bbl@save@sfcodes{\bbl@fs@chars}%
1651 \def\bbl@post@fs{%
1652   \bbl@save@sfcodes
1653   \edef\bbl@tempa{\bbl@cl{frspc}}%
1654   \edef\bbl@tempa{\expandafter\@car\bbl@tempa\@nil}%
1655   \if u\bbl@tempa      % do nothing
1656   \else\if n\bbl@tempa  % non french
1657     \def\bbl@elt##1##2##3{%
1658       \ifnum\sfcode`##1=##2\relax
1659         \babel@savevariable{\sfcode`##1}%
1660         \sfcode`##1=##3\relax
1661       \fi}%
1662     \bbl@fs@chars
1663   \else\if y\bbl@tempa  % french
1664     \def\bbl@elt##1##2##3{%
1665       \ifnum\sfcode`##1=##3\relax
1666         \babel@savevariable{\sfcode`##1}%
1667         \sfcode`##1=##2\relax
1668       \fi}%
1669     \bbl@fs@chars
1670   \fi\fi\fi}

```

## 8.8 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.

```

1671 \bbl@trace{Short tags}
1672 \def\babeltags#1{%
1673   \edef\bbl@tempa{\zap@space#1 \@empty}%
1674   \def\bbl@tempb##1=##2\@{#1}%
1675   \edef\bbl@tempc{%
1676     \noexpand\newcommand
1677     \expandafter\noexpand\csname ##1\endcsname{%
1678       \noexpand\protect
1679       \expandafter\noexpand\csname otherlanguage*\endcsname{##2}}
1680     \noexpand\newcommand
1681     \expandafter\noexpand\csname text##1\endcsname{%

```



```

1682 \noexpand\foreignlanguage{##2}}
1683 \bbl@tempc}%
1684 \bbl@for\bbl@tempa\bbl@tempa{%
1685 \expandafter\bbl@tempb\bbl@tempa\@@}}

```

## 8.9 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.

```

1686 \bbl@trace{Hyphens}
1687 \@onlypreamble\babelhyphenation
1688 \AtEndOfPackage{%
1689 \newcommand\babelhyphenation[2][\@empty]{%
1690 \ifx\bbl@hyphenation@ \relax
1691 \let\bbl@hyphenation@ \@empty
1692 \fi
1693 \ifx\bbl@hyphlist \@empty \else
1694 \bbl@warning{%
1695 You must not intermingle \string\selectlanguage\space and\\
1696 \string\babelhyphenation\space or some exceptions will not\\
1697 be taken into account. Reported}%
1698 \fi
1699 \ifx\@empty#1%
1700 \protected@edef\bbl@hyphenation@{\bbl@hyphenation@\space#2}%
1701 \else
1702 \bbl@vforeach{#1}{%
1703 \def\bbl@tempa{##1}%
1704 \bbl@fixname\bbl@tempa
1705 \bbl@iflanguage\bbl@tempa{%
1706 \bbl@csarg\protected@edef{hyphenation@\bbl@tempa}{%
1707 \bbl@ifunset{bbl@hyphenation@\bbl@tempa}%
1708 {}%
1709 {\csname bbl@hyphenation@\bbl@tempa\endcsname\space}%
1710 #2}}}%
1711 \fi}}

```

`\bbl@allowhyphens` This macro makes hyphenation possible. Basically its definition is nothing more than `\nobreak` `\hskip 0pt` plus `Opt`<sup>32</sup>.

```

1712 \def\bbl@allowhyphens{\ifvmode\else\nobreak\hskip\z@skip\fi}
1713 \def\bbl@t@one{T1}
1714 \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`.

```

1715 \newcommand\babellnullhyphen{\char\hyphenchar\font}
1716 \def\babelhyphen{\active@prefix\babelhyphen\bbl@hyphen}
1717 \def\bbl@hyphen{%
1718 \@ifstar{\bbl@hyphen@i @}{\bbl@hyphen@i \@empty}}
1719 \def\bbl@hyphen@i#1#2{%
1720 \bbl@ifunset{bbl@hy@#1#2\@empty}%
1721 {\csname bbl@#1usehyphen\endcsname{\discretionary{#2}{}{#2}}}%
1722 {\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

<sup>32</sup> $\TeX$  begins and ends a word for hyphenation at a glue node. The penalty prevents a linebreak at this glue node.

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.

```
1723 \def\bbl@usehyphen#1{%
1724   \leavevmode
1725   \ifdim\lastskip>\z@\mbox{#1}\else\nobreak#1\fi
1726   \nobreak\hskip\z@skip}
1727 \def\bbl@usehyphen#1{%
1728   \leavevmode\ifdim\lastskip>\z@\mbox{#1}\else#1\fi}
```

The following macro inserts the hyphen char.

```
1729 \def\bbl@hyphenchar{%
1730   \ifnum\hyphenchar\font=\m@ne
1731     \babe\nullhyphen
1732   \else
1733     \char\hyphenchar\font
1734   \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.

```
1735 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
1736 \def\bbl@hy@soft{\bbl@usehyphen{\discretionary{\bbl@hyphenchar}{}}{}}
1737 \def\bbl@hy@hard{\bbl@usehyphen\bbl@hyphenchar}
1738 \def\bbl@hy@@hard{\bbl@usehyphen\bbl@hyphenchar}
1739 \def\bbl@hy@nobreak{\bbl@usehyphen{\mbox{\bbl@hyphenchar}}{}}
1740 \def\bbl@hy@nobreak{\mbox{\bbl@hyphenchar}}
1741 \def\bbl@hy@repeat{%
1742   \bbl@usehyphen%
1743   \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}
1744 \def\bbl@hy@repeat{%
1745   \bbl@usehyphen%
1746   \discretionary{\bbl@hyphenchar}{\bbl@hyphenchar}{\bbl@hyphenchar}}
1747 \def\bbl@hy@empty{\hskip\z@skip}
1748 \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.

```
1749 \def\bbl@disc#1#2{\nobreak\discretionary{#2-}{#1}\bbl@allowhyphens}
```

## 8.10 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.

```
1750 \bbl@trace{Multiencoding strings}
1751 \def\bbl@tglobal#1{\global\let#1#1}
1752 \def\bbl@recatcode#1{% TODO. Used only once?
1753   \@tempcnta="7F
1754   \def\bbl@tempa{%
1755     \ifnum\@tempcnta>"FF\else
1756       \catcode\@tempcnta=#1\relax
1757       \advance\@tempcnta\@ne
1758       \expandafter\bbl@tempa
```

```

1759 \fi}%
1760 \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).

```

1761 \@ifpackagewith{babel}{nocase}%
1762 {\let\bbl@patchuclc\relax}%
1763 {\def\bbl@patchuclc{%
1764   \global\let\bbl@patchuclc\relax
1765   \g@addto@macro\@uclclist{\reserved@b{\reserved@b\bbl@uclc}}%
1766   \gdef\bbl@uclc##1{%
1767     \let\bbl@encoded\bbl@encoded@uclc
1768     \bbl@ifunset{\language @bbl@uclc}% and resumes it
1769     {##1}%
1770     {\let\bbl@tempa##1\relax % Used by LANG@bbl@uclc
1771       \csname\language @bbl@uclc\endcsname}%
1772     {\bbl@tolower\@empty}{\bbl@toupper\@empty}}%
1773   \gdef\bbl@tolower{\csname\language @bbl@lc\endcsname}%
1774   \gdef\bbl@toupper{\csname\language @bbl@uc\endcsname}}}
1775 <<More package options>> ≡
1776 \DeclareOption{nocase}{}
1777 <</More package options>>

```

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

```

1778 <<More package options>> ≡
1779 \let\bbl@opt@strings\@nnil % accept strings=value
1780 \DeclareOption{strings}{\def\bbl@opt@strings{\BabelStringsDefault}}
1781 \DeclareOption{strings=encoded}{\let\bbl@opt@strings\relax}
1782 \def\BabelStringsDefault{generic}
1783 <</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.

```

1784 \@onlypreamble\StartBabelCommands
1785 \def\StartBabelCommands{%
1786   \begingroup
1787   \bbl@recatcode{11}%
1788   <<Macros local to BabelCommands>>
1789   \def\bbl@provstring##1##2{%
1790     \providecommand##1{##2}%
1791     \bbl@tglobal##1}%
1792   \global\let\bbl@scafter\@empty
1793   \let\StartBabelCommands\bbl@startcmds
1794   \ifx\BabelLanguages\relax
1795     \let\BabelLanguages\CurrentOption
1796   \fi
1797   \begingroup

```

```

1798 \let\bbl@screset\@nnil % local flag - disable 1st stopcommands
1799 \StartBabelCommands}
1800 \def\bbl@startcmds{%
1801   \ifx\bbl@screset\@nnil\else
1802     \bbl@usehooks{stopcommands}{}%
1803   \fi
1804 \endgroup
1805 \begingroup
1806 \@ifstar
1807   {\ifx\bbl@opt@strings\@nnil
1808     \let\bbl@opt@strings\BabelStringsDefault
1809   \fi
1810   \bbl@startcmds@i}%
1811   \bbl@startcmds@i}
1812 \def\bbl@startcmds@i#1#2{%
1813   \edef\bbl@L{\zap@space#1 \@empty}%
1814   \edef\bbl@G{\zap@space#2 \@empty}%
1815   \bbl@startcmds@ii}
1816 \let\bbl@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.

```

1817 \newcommand\bbl@startcmds@ii[1][\@empty]{%
1818   \let\SetString\@gobbletwo
1819   \let\bbl@stringdef\@gobbletwo
1820   \let\AfterBabelCommands\@gobble
1821   \ifx\@empty#1%
1822     \def\bbl@sc@label{generic}%
1823     \def\bbl@encstring##1##2{%
1824       \ProvideTextCommandDefault##1{##2}%
1825       \bbl@toglobal##1%
1826       \expandafter\bbl@toglobal\csname\string?\string##1\endcsname}%
1827     \let\bbl@sctest\in@true
1828   \else
1829     \let\bbl@sc@charset\space % <- zapped below
1830     \let\bbl@sc@fontenc\space % <- " "
1831     \def\bbl@tempa##1=##2\@nil{%
1832       \bbl@csarg\edef{sc@\zap@space##1 \@empty}{##2 }}%
1833     \bbl@vforeach{label=#1}{\bbl@tempa##1\@nil}%
1834     \def\bbl@tempa##1 ##2{% space -> comma
1835       ##1%
1836       \ifx\@empty##2\else\ifx,##1,\else,\fi\bbl@afterfi\bbl@tempa##2\fi}%
1837     \edef\bbl@sc@fontenc{\expandafter\bbl@tempa\bbl@sc@fontenc\@empty}%
1838     \edef\bbl@sc@label{\expandafter\zap@space\bbl@sc@label\@empty}%
1839     \edef\bbl@sc@charset{\expandafter\zap@space\bbl@sc@charset\@empty}%
1840     \def\bbl@encstring##1##2{%
1841       \bbl@foreach\bbl@sc@fontenc{%
1842         \bbl@ifunset{T@###1}%
1843         {}%
1844         {\ProvideTextCommand##1{####1}{##2}%
1845         \bbl@toglobal##1%
1846         \expandafter

```

```

1847         \bbl@toglobal\csname####1\string##1\endcsname}}}%
1848     \def\bbl@sctest{%
1849         \bbl@xin@{\, \bbl@opt@strings,}{\, \bbl@sc@label, \bbl@sc@fontenc,}}%
1850     \fi
1851     \ifx\bbl@opt@strings\@nnil          % ie, no strings key -> defaults
1852     \else\ifx\bbl@opt@strings\relax      % ie, strings=encoded
1853         \let\AfterBabelCommands\bbl@aftercmds
1854         \let\SetString\bbl@setstring
1855         \let\bbl@stringdef\bbl@encstring
1856     \else          % ie, strings=value
1857     \bbl@sctest
1858     \ifin@
1859         \let\AfterBabelCommands\bbl@aftercmds
1860         \let\SetString\bbl@setstring
1861         \let\bbl@stringdef\bbl@provstring
1862     \fi\fi\fi
1863     \bbl@scswitch
1864     \ifx\bbl@G\@empty
1865         \def\SetString##1##2{%
1866             \bbl@error{Missing group for string \string##1}%
1867             {You must assign strings to some category, typically\\%
1868             captions or extras, but you set none}}%
1869     \fi
1870     \ifx\@empty#1%
1871         \bbl@usehooks{defaultcommands}}}%
1872     \else
1873         \@expandtwoargs
1874         \bbl@usehooks{encodedcommands}{\bbl@sc@charset}{\bbl@sc@fontenc}}}%
1875     \fi}

```

There are two versions of `\bbl@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 (`\bbl@screset` is used to keep track of this). The second version is used in the preamble and packages loaded after `babel` and does nothing. The macro `\bbl@forlang` loops `\bbl@L` but its body is executed only if the value is in `\BabelLanguages` (inside `babel`) or `\date \langle language \rangle` is defined (after `babel` has been loaded). There are also two version of `\bbl@forlang`. The first one skips the current iteration if the language is not in `\BabelLanguages` (used in `ldfs`), and the second one skips undefined languages (after `babel` has been loaded).

```

1876 \def\bbl@forlang#1#2{%
1877     \bbl@for#1\bbl@L{%
1878         \bbl@xin@{\, #1,}{\, \BabelLanguages,}%
1879         \ifin@#2\relax\fi}}
1880 \def\bbl@scswitch{%
1881     \bbl@forlang\bbl@tempa{%
1882         \ifx\bbl@G\@empty\else
1883             \ifx\SetString\@gobbletwo\else
1884                 \edef\bbl@GL{\bbl@G\bbl@tempa}%
1885                 \bbl@xin@{\, \bbl@GL,}{\, \bbl@screset,}%
1886             \ifin@\else
1887                 \global\expandafter\let\csname\bbl@GL\endcsname\@undefined
1888                 \xdef\bbl@screset{\bbl@screset, \bbl@GL}%
1889             \fi
1890         \fi
1891     \fi}}
1892 \AtEndOfPackage{%
1893     \def\bbl@forlang#1#2{\bbl@for#1\bbl@L{\bbl@ifunset{date#1}}{\, #2}}}%
1894     \let\bbl@scswitch\relax}
1895 \onlypreamble\EndBabelCommands
1896 \def\EndBabelCommands{%

```

```

1897 \bbl@usehooks{stopcommands}{}%
1898 \endgroup
1899 \endgroup
1900 \bbl@scafter}
1901 \let\bbl@endcommands\EndBabelCommands

```

Now we define commands to be used inside \StartBabelCommands.

**Strings** The following macro is the actual definition of \SetString when it is “active” First save the “switcher”. Create it if undefined. Strings are defined only if undefined (ie, like \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.

```

1902 \def\bbl@setstring#1#2{% eg, \prefacename{<string>}
1903 \bbl@forlang\bbl@tempa{%
1904 \edef\bbl@LC{\bbl@tempa\bbl@stripslash#1}%
1905 \bbl@ifunset{\bbl@LC}% eg, \germanchaptername
1906 {\bbl@exp{%
1907 \global\bbbl@add\<\bbl@G\bbl@tempa>\bbbl@scset\#1\<\bbl@LC>}}}%
1908 }%
1909 \def\BabelString{#2}%
1910 \bbl@usehooks{stringprocess}{}%
1911 \expandafter\bbl@stringdef
1912 \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.

```

1913 \ifx\bbl@opt@strings\relax
1914 \def\bbl@scset#1#2{\def#1{\bbl@encoded#2}}
1915 \bbl@patchuclc
1916 \let\bbl@encoded\relax
1917 \def\bbl@encoded@uclc#1{%
1918 \@inmathwarn#1%
1919 \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
1920 \expandafter\ifx\csname ?\string#1\endcsname\relax
1921 \TextSymbolUnavailable#1%
1922 \else
1923 \csname ?\string#1\endcsname
1924 \fi
1925 \else
1926 \csname\cf@encoding\string#1\endcsname
1927 \fi}
1928 \else
1929 \def\bbl@scset#1#2{\def#1{#2}}
1930 \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.

```

1931 <<*Macros local to BabelCommands>> ≡
1932 \def\SetStringLoop##1##2{%
1933 \def\bbl@templ####1{\expandafter\noexpand\csname##1\endcsname}%
1934 \count@\z@
1935 \bbl@loop\bbl@tempa{##2}{% empty items and spaces are ok
1936 \advance\count@\@ne
1937 \toks@\expandafter{\bbl@tempa}%
1938 \bbl@exp{%
1939 \SetString\bbl@templ{\romannumeral\count@}{\the\toks@}%

```

```

1940         \count@=\the\count@\relax}}}%
1941 <</Macros local to BabelCommands>>

```

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

```

1942 \def\bbl@aftercmds#1{%
1943   \toks@\expandafter{\bbl@scafter#1}%
1944   \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.

```

1945 <<*Macros local to BabelCommands>> ≡
1946   \newcommand\SetCase[3][1]{%
1947     \bbl@patchuclc
1948     \bbl@forlang\bbl@tempa{%
1949       \expandafter\bbl@encstring
1950       \csname\bbl@tempa @bbl@uclc\endcsname{\bbl@tempa##1}%
1951       \expandafter\bbl@encstring
1952       \csname\bbl@tempa @bbl@uc\endcsname{##2}%
1953       \expandafter\bbl@encstring
1954       \csname\bbl@tempa @bbl@lc\endcsname{##3}}}%
1955 <</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.

```

1956 <<*Macros local to BabelCommands>> ≡
1957   \newcommand\SetHyphenMap[1]{%
1958     \bbl@forlang\bbl@tempa{%
1959       \expandafter\bbl@stringdef
1960       \csname\bbl@tempa @bbl@hyphenmap\endcsname{##1}}}%
1961 <</Macros local to BabelCommands>>

```

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

```

1962 \newcommand\BabelLower[2]{% one to one.
1963   \ifnum\lccode#1=#2\else
1964     \babel@savevariable{\lccode#1}%
1965     \lccode#1=#2\relax
1966   \fi}
1967 \newcommand\BabelLowerMM[4]{% many-to-many
1968   \@tempcnta=#1\relax
1969   \@tempcntb=#4\relax
1970   \def\bbl@tempa{%
1971     \ifnum\@tempcnta>#2\else
1972       \@expandtwoargs\BabelLower{\the\@tempcnta}{\the\@tempcntb}%
1973       \advance\@tempcnta#3\relax
1974       \advance\@tempcntb#3\relax
1975       \expandafter\bbl@tempa
1976     \fi}%
1977   \bbl@tempa}
1978 \newcommand\BabelLowerMO[4]{% many-to-one
1979   \@tempcnta=#1\relax
1980   \def\bbl@tempa{%
1981     \ifnum\@tempcnta>#2\else
1982       \@expandtwoargs\BabelLower{\the\@tempcnta}{#4}%
1983       \advance\@tempcnta#3
1984       \expandafter\bbl@tempa
1985     \fi}%

```

```
1986 \bbl@tempa}
```

The following package options control the behavior of hyphenation mapping.

```
1987 <<*More package options>> ≡
1988 \DeclareOption{hyphenmap=off}{\chardef\bbl@opt@hyphenmap\z@}
1989 \DeclareOption{hyphenmap=first}{\chardef\bbl@opt@hyphenmap@ne}
1990 \DeclareOption{hyphenmap=select}{\chardef\bbl@opt@hyphenmap\tw@}
1991 \DeclareOption{hyphenmap=other}{\chardef\bbl@opt@hyphenmap\thr@@}
1992 \DeclareOption{hyphenmap=other*}{\chardef\bbl@opt@hyphenmap4\relax}
1993 <</More package options>>
```

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

```
1994 \AtEndOfPackage{%
1995   \ifx\bbl@opt@hyphenmap\undefined
1996     \bbl@xin@{,}{\bbl@language@opts}%
1997     \chardef\bbl@opt@hyphenmap\ifin4\else\ne\fi
1998   \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.

```
1999 \newcommand\setlocalecaption{% TODO. Catch typos. What about ensure?
2000   \@ifstar\bbl@setcaption@s\bbl@setcaption@x}
2001 \def\bbl@setcaption@x#1#2#3{% language caption-name string
2002   \bbl@trim@def\bbl@tempa{#2}%
2003   \bbl@xin@{.template}{\bbl@tempa}%
2004   \ifin@
2005     \bbl@ini@captions@template{#3}{#1}%
2006   \else
2007     \edef\bbl@tempd{%
2008       \expandafter\expandafter\expandafter
2009       \strip@prefix\expandafter\meaning\csname captions#1\endcsname}%
2010     \bbl@xin@
2011       {\expandafter\string\csname #2name\endcsname}%
2012     {\bbl@tempd}%
2013   \ifin@ % Renew caption
2014     \bbl@xin@{\string\bbl@scset}{\bbl@tempd}%
2015     \ifin@
2016       \bbl@exp{%
2017         \\bbl@ifsamestring{\bbl@tempa}{\language}%
2018         {\\bbl@scset\<#2name>\<#1#2name>}%
2019         {}}%
2020       \else % Old way converts to new way
2021         \bbl@ifunset{#1#2name}%
2022         {\bbl@exp{%
2023           \\bbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}%
2024           \\bbl@ifsamestring{\bbl@tempa}{\language}%
2025           {\def\<#2name>{\<#1#2name>}}%
2026           {}}}%
2027         {}%
2028       \fi
2029     \else
2030       \bbl@xin@{\string\bbl@scset}{\bbl@tempd}% New
2031       \ifin@ % New way
2032         \bbl@exp{%
2033           \\bbl@add\<captions#1>{\\bbl@scset\<#2name>\<#1#2name>}%
2034           \\bbl@ifsamestring{\bbl@tempa}{\language}%
2035           {\\bbl@scset\<#2name>\<#1#2name>}%
2036           {}}%
```



```

2037 \else % Old way, but defined in the new way
2038 \bbl@exp{%
2039 \\\bbl@add\<captions#1>{\def\<#2name>{\<#1#2name>}}%
2040 \\\bbl@ifsamestring{\bbl@tempa}{\language}%
2041 {\def\<#2name>{\<#1#2name>}}%
2042 }%
2043 \fi%
2044 \fi
2045 \@namedef{#1#2name}{#3}%
2046 \toks@\expandafter{\bbl@captionslist}%
2047 \bbl@exp{\in{\<#2name>}{\the\toks@}}%
2048 \ifin@else
2049 \bbl@exp{\\\bbl@add\\bbl@captionslist{\<#2name>}}%
2050 \bbl@tglobal\bbl@captionslist
2051 \fi
2052 \fi}
2053 % \def\bbl@setcaption@s#1#2#3{} % TODO. Not yet implemented

```

## 8.11 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.

```

2054 \bbl@trace{Macros related to glyphs}
2055 \def\set@low@box#1{\setbox\tw@{\hbox{,}}\setbox\z@{\hbox{#1}}%
2056 \dimen\z@{\ht\z@ \advance\dimen\z@ -\ht\tw@}%
2057 \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.

```

2058 \def\save@sf@q#1{\leavevmode
2059 \begingroup
2060 \edef\@SF{\spacefactor\the\spacefactor}#1\@SF
2061 \endgroup}

```

## 8.12 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`.

### 8.12.1 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.

```

2062 \ProvideTextCommand{\quotedblbase}{OT1}{%
2063 \save@sf@q{\set@low@box{\textquotedblright\}}%
2064 \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.

```

2065 \ProvideTextCommandDefault{\quotedblbase}{%
2066 \UseTextSymbol{OT1}{\quotedblbase}}

```

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

```

2067 \ProvideTextCommand{\quotesinglbase}{OT1}{%
2068 \save@sf@q{\set@low@box{\textquoteright\}}%
2069 \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.

```

2070 \ProvideTextCommandDefault{\quotesinglbase}{%
2071 \UseTextSymbol{OT1}{\quotesinglbase}}

```

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

```

2072 \ProvideTextCommand{\guillemetleft}{OT1}{%
2073   \ifmmode
2074     \ll
2075   \else
2076     \save@sf@q{\nobreak
2077       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2078   \fi}
2079 \ProvideTextCommand{\guillemetright}{OT1}{%
2080   \ifmmode
2081     \gg
2082   \else
2083     \save@sf@q{\nobreak
2084       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2085   \fi}
2086 \ProvideTextCommand{\guillemotleft}{OT1}{%
2087   \ifmmode
2088     \ll
2089   \else
2090     \save@sf@q{\nobreak
2091       \raise.2ex\hbox{$\scriptscriptstyle\ll$}\bbl@allowhyphens}%
2092   \fi}
2093 \ProvideTextCommand{\guillemotright}{OT1}{%
2094   \ifmmode
2095     \gg
2096   \else
2097     \save@sf@q{\nobreak
2098       \raise.2ex\hbox{$\scriptscriptstyle\gg$}\bbl@allowhyphens}%
2099   \fi}

```

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

```

2100 \ProvideTextCommandDefault{\guillemetleft}{%
2101   \UseTextSymbol{OT1}{\guillemetleft}}
2102 \ProvideTextCommandDefault{\guillemetright}{%
2103   \UseTextSymbol{OT1}{\guillemetright}}
2104 \ProvideTextCommandDefault{\guillemotleft}{%
2105   \UseTextSymbol{OT1}{\guillemotleft}}
2106 \ProvideTextCommandDefault{\guillemotright}{%
2107   \UseTextSymbol{OT1}{\guillemotright}}

```

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

```

2108 \ProvideTextCommand{\guilsinglleft}{OT1}{%
2109   \ifmmode
2110     <%
2111   \else
2112     \save@sf@q{\nobreak
2113       \raise.2ex\hbox{$\scriptscriptstyle<$}\bbl@allowhyphens}%
2114   \fi}
2115 \ProvideTextCommand{\guilsinglright}{OT1}{%
2116   \ifmmode
2117     >%
2118   \else
2119     \save@sf@q{\nobreak
2120       \raise.2ex\hbox{$\scriptscriptstyle>$}\bbl@allowhyphens}%
2121   \fi}

```

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

```

2122 \ProvideTextCommandDefault{\guilsinglleft}{%
2123   \UseTextSymbol{OT1}{\guilsinglleft}}
2124 \ProvideTextCommandDefault{\guilsinglright}{%
2125   \UseTextSymbol{OT1}{\guilsinglright}}

```

### 8.12.2 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.

```

2126 \DeclareTextCommand{\ij}{OT1}{%
2127   i\kern-0.02em\bbl@allowhyphens j}
2128 \DeclareTextCommand{\IJ}{OT1}{%
2129   I\kern-0.02em\bbl@allowhyphens J}
2130 \DeclareTextCommand{\ij}{T1}{\char188}
2131 \DeclareTextCommand{\IJ}{T1}{\char156}

```

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

```

2132 \ProvideTextCommandDefault{\ij}{%
2133   \UseTextSymbol{OT1}{\ij}}
2134 \ProvideTextCommandDefault{\IJ}{%
2135   \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).

```

2136 \def\crrtic@{\hrule height0.1ex width0.3em}
2137 \def\crttic@{\hrule height0.1ex width0.33em}
2138 \def\ddj@{%
2139   \setbox0\hbox{d}\dimen@=\ht0
2140   \advance\dimen@1ex
2141   \dimen@.45\dimen@
2142   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2143   \advance\dimen@ii.5ex
2144   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crrtic@}}}}
2145 \def\DDJ@{%
2146   \setbox0\hbox{D}\dimen@=.55\ht0
2147   \dimen@ii\expandafter\rem@pt\the\fontdimen\@ne\font\dimen@
2148   \advance\dimen@ii.15ex % correction for the dash position
2149   \advance\dimen@ii-.15\fontdimen7\font % correction for cmtt font
2150   \dimen\thr@@\expandafter\rem@pt\the\fontdimen7\font\dimen@
2151   \leavevmode\rlap{\raise\dimen@\hbox{\kern\dimen@ii\vbox{\crttic@}}}}
2152 %
2153 \DeclareTextCommand{\dj}{OT1}{\ddj@ d}
2154 \DeclareTextCommand{\DJ}{OT1}{\DDJ@ D}

```

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

```

2155 \ProvideTextCommandDefault{\dj}{%
2156   \UseTextSymbol{OT1}{\dj}}
2157 \ProvideTextCommandDefault{\DJ}{%
2158   \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.

```

2159 \DeclareTextCommand{\SS}{OT1}{\SS}
2160 \ProvideTextCommandDefault{\SS}{\UseTextSymbol{OT1}{\SS}}

```

### 8.12.3 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 2161 \ProvideTextCommandDefault{\glq}{%
      2162 \textormath{\quotesinglbase}{\mbox{\quotesinglbase}}}
```

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

```
2163 \ProvideTextCommand{\grq}{T1}{%
2164 \textormath{\kern\z@\textquoteleft}{\mbox{\textquoteleft}}}
2165 \ProvideTextCommand{\grq}{TU}{%
2166 \textormath{\textquoteleft}{\mbox{\textquoteleft}}}
2167 \ProvideTextCommand{\grq}{OT1}{%
2168 \save@sf@q{\kern-.0125em
2169 \textormath{\textquoteleft}{\mbox{\textquoteleft}}}%
2170 \kern.07em\relax}}
2171 \ProvideTextCommandDefault{\grq}{\UseTextSymbol{OT1}\grq}
```

`\glqq` The ‘german’ double quotes.

```
\grqq 2172 \ProvideTextCommandDefault{\glqq}{%
      2173 \textormath{\quotedblbase}{\mbox{\quotedblbase}}}
```

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

```
2174 \ProvideTextCommand{\grqq}{T1}{%
2175 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2176 \ProvideTextCommand{\grqq}{TU}{%
2177 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}
2178 \ProvideTextCommand{\grqq}{OT1}{%
2179 \save@sf@q{\kern-.07em
2180 \textormath{\textquotedblleft}{\mbox{\textquotedblleft}}}%
2181 \kern.07em\relax}}
2182 \ProvideTextCommandDefault{\grqq}{\UseTextSymbol{OT1}\grqq}
```

`\flq` The ‘french’ single guillemets.

```
\frq 2183 \ProvideTextCommandDefault{\flq}{%
      2184 \textormath{\guilsinglleft}{\mbox{\guilsinglleft}}}
2185 \ProvideTextCommandDefault{\frq}{%
2186 \textormath{\guilsinglright}{\mbox{\guilsinglright}}}
```

`\flqq` The ‘french’ double guillemets.

```
\frqq 2187 \ProvideTextCommandDefault{\flqq}{%
      2188 \textormath{\guillemetleft}{\mbox{\guillemetleft}}}
2189 \ProvideTextCommandDefault{\frqq}{%
2190 \textormath{\guillemetright}{\mbox{\guillemetright}}}
```

### 8.12.4 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).

```
2191 \def\umlauthigh{%
```

```

2192 \def\bbl@umlauta##1{\leavevmode\bgroup%
2193   \expandafter\accent\csname\fontencoding dqpos\endcsname
2194   ##1\bbl@allowhyphens\egroup}%
2195 \let\bbl@umlaute\bbl@umlauta}
2196 \def\umlautlow{%
2197   \def\bbl@umlauta{\protect\lower@umlaut}}
2198 \def\umlautelow{%
2199   \def\bbl@umlaute{\protect\lower@umlaut}}
2200 \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.

```

2201 \expandafter\ifx\csname U@D\endcsname\relax
2202   \csname newdimen\endcsname\U@D
2203 \fi

```

The following code fools  $\TeX$ 's `make_accent` procedure about the current x-height of the font to force another placement of the umlaut character. First we have to save the current x-height of the font, because we'll change this font dimension and this is always done globally. Then we compute the new x-height in such a way that the umlaut character is lowered to the base character. The value of `.45ex` depends on the METAFONT parameters with which the fonts were built. (Just try out, which value will look best.) If the new x-height is too low, it is not changed. Finally we call the `\accent` primitive, reset the old x-height and insert the base character in the argument.

```

2204 \def\lower@umlaut#1{%
2205   \leavevmode\bgroup
2206   \U@D 1ex%
2207   {\setbox\z@\hbox{%
2208     \expandafter\char\csname\fontencoding dqpos\endcsname}%
2209     \dimen@ -.45ex\advance\dimen@\ht\z@
2210     \ifdim 1ex<\dimen@ \fontdimen5\font\dimen@ \fi}%
2211   \expandafter\accent\csname\fontencoding dqpos\endcsname
2212   \fontdimen5\font\U@D #1%
2213 \egroup}

```

For all vowels we declare `\` to be a composite command which uses `\bbl@umlauta` or `\bbl@umlaute` to position the umlaut character. We need to be sure that these definitions override the ones that are provided when the package `fontenc` with option `OT1` is used. Therefore these declarations are postponed until the beginning of the document. Note these definitions only apply to some languages, but `babel` sets them for *all* languages – you may want to redefine `\bbl@umlauta` and/or `\bbl@umlaute` for a language in the corresponding `ldf` (using the `babel` switching mechanism, of course).

```

2214 \AtBeginDocument{%
2215   \DeclareTextCompositeCommand{\}{OT1}{a}{\bbl@umlauta{a}}%
2216   \DeclareTextCompositeCommand{\}{OT1}{e}{\bbl@umlaute{e}}%
2217   \DeclareTextCompositeCommand{\}{OT1}{i}{\bbl@umlaute{i}}%
2218   \DeclareTextCompositeCommand{\}{OT1}{\i}{\bbl@umlaute{i}}%
2219   \DeclareTextCompositeCommand{\}{OT1}{o}{\bbl@umlauta{o}}%
2220   \DeclareTextCompositeCommand{\}{OT1}{u}{\bbl@umlauta{u}}%
2221   \DeclareTextCompositeCommand{\}{OT1}{A}{\bbl@umlauta{A}}%
2222   \DeclareTextCompositeCommand{\}{OT1}{E}{\bbl@umlaute{E}}%
2223   \DeclareTextCompositeCommand{\}{OT1}{I}{\bbl@umlaute{I}}%
2224   \DeclareTextCompositeCommand{\}{OT1}{O}{\bbl@umlauta{O}}%
2225   \DeclareTextCompositeCommand{\}{OT1}{U}{\bbl@umlauta{U}}%

```

Finally, make sure the default hyphenrules are defined (even if empty). For internal use, another empty `\language` is defined. Currently used in `Amharic`.

```

2226 \ifx\l@english\undefined
2227   \chardef\l@english\z@
2228 \fi

```

```

2229% The following is used to cancel rules in ini files (see Amharic).
2230 \ifx\l@unhyphenated\undefined
2231 \newlanguage\l@unhyphenated
2232 \fi

```

## 8.13 Layout

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

```

2233 \bbl@trace{Bidi layout}
2234 \providecommand\IfBabelLayout[3]{#3}%
2235 \newcommand\BabelPatchSection[1]{%
2236   \@ifundefined{#1}{}{%
2237     \bbl@exp{\let\<bbl@ss@#1>\<#1>}%
2238     \@namedef{#1}{%
2239       \ifstar\bbl@presec@s{#1}%
2240       {\@dblarg\bbl@presec@x{#1}}}%
2241 \def\bbl@presec@x#1[#2]#3{%
2242   \bbl@exp{%
2243     \\select@language@x{\bbl@main@language}%
2244     \\bbl@cs{sspre@#1}%
2245     \\bbl@cs{ss@#1}%
2246     [\\foreignlanguage{\language}{\unexpanded{#2}}]%
2247     {\\foreignlanguage{\language}{\unexpanded{#3}}}%
2248     \\select@language@x{\language}}%
2249 \def\bbl@presec@s#1#2{%
2250   \bbl@exp{%
2251     \\select@language@x{\bbl@main@language}%
2252     \\bbl@cs{sspre@#1}%
2253     \\bbl@cs{ss@#1}*%
2254     {\\foreignlanguage{\language}{\unexpanded{#2}}}%
2255     \\select@language@x{\language}}%
2256 \IfBabelLayout{sectioning}%
2257   {\BabelPatchSection{part}%
2258    \BabelPatchSection{chapter}%
2259    \BabelPatchSection{section}%
2260    \BabelPatchSection{subsection}%
2261    \BabelPatchSection{subsubsection}%
2262    \BabelPatchSection{paragraph}%
2263    \BabelPatchSection{subparagraph}%
2264    \def\babel@toc#1{%
2265      \select@language@x{\bbl@main@language}}}%
2266 \IfBabelLayout{captions}%
2267   {\BabelPatchSection{caption}}%

```

## 8.14 Load engine specific macros

```

2268 \bbl@trace{Input engine specific macros}
2269 \ifcase\bbl@engine
2270   \input txtbabel.def
2271 \or
2272   \input luababel.def
2273 \or
2274   \input xebabel.def
2275 \fi

```

## 8.15 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.

```
2276 \bbl@trace{Creating languages and reading ini files}
2277 \let\bbl@extend@ini@gobble
2278 \newcommand\babelprovide[2][{}%
2279   \let\bbl@savelangname\language
2280   \edef\bbl@savelocaleid{\the\localeid}%
2281   % Set name and locale id
2282   \edef\language{#2}%
2283   \bbl@id@assign
2284   % Initialize keys
2285   \let\bbl@KVP@captions\@nil
2286   \let\bbl@KVP@date\@nil
2287   \let\bbl@KVP@import\@nil
2288   \let\bbl@KVP@main\@nil
2289   \let\bbl@KVP@script\@nil
2290   \let\bbl@KVP@language\@nil
2291   \let\bbl@KVP@hyphenrules\@nil
2292   \let\bbl@KVP@linebreaking\@nil
2293   \let\bbl@KVP@justification\@nil
2294   \let\bbl@KVP@mapfont\@nil
2295   \let\bbl@KVP@maparabic\@nil
2296   \let\bbl@KVP@mapdigits\@nil
2297   \let\bbl@KVP@intraspace\@nil
2298   \let\bbl@KVP@intrapenalty\@nil
2299   \let\bbl@KVP@onchar\@nil
2300   \let\bbl@KVP@transforms\@nil
2301   \global\let\bbl@release@transforms\@empty
2302   \let\bbl@KVP@alph\@nil
2303   \let\bbl@KVP@Alph\@nil
2304   \let\bbl@KVP@labels\@nil
2305   \bbl@csarg\let{KVP@labels*}\@nil
2306   \global\let\bbl@inidata\@empty
2307   \global\let\bbl@extend@ini@gobble
2308   \gdef\bbl@key@list{}%
2309   \bbl@forkv{#1}{% TODO - error handling
2310     \in@{/{}}{##1}%
2311     \ifin@
2312       \global\let\bbl@extend@ini\bbl@extend@ini@aux
2313       \bbl@renewinikey##1\@{##2}%
2314     \else
2315       \bbl@csarg\def{KVP@##1}{##2}%
2316     \fi}%
2317   \chardef\bbl@howloaded=% 0:none; 1:ldf without ini; 2:ini
2318   \bbl@ifunset{date#2}\z@{\bbl@ifunset{\bbl@llevel#2}\@ne\tw@}%
2319   % == init ==
2320   \ifx\bbl@screset\@undefined
2321     \bbl@ldfinit
2322   \fi
2323   % ==
2324   \let\bbl@lbkflag\relax % \@empty = do setup linebreak
2325   \ifcase\bbl@howloaded
2326     \let\bbl@lbkflag\@empty % new
2327   \else
2328     \ifx\bbl@KVP@hyphenrules\@nil\else
2329       \let\bbl@lbkflag\@empty
2330     \fi
2331     \ifx\bbl@KVP@import\@nil\else
2332       \let\bbl@lbkflag\@empty
```

```

2333 \fi
2334 \fi
2335 % == import, captions ==
2336 \ifx\bbbl@KVP@import\@nil\else
2337 \bbbl@exp{\bbbl@ifblank{\bbbl@KVP@import}}%
2338 {\ifx\bbbl@initoload\relax
2339 \begingroup
2340 \def\BabelBeforeIni##1##2{\gdef\bbbl@KVP@import{##1}\endinput}%
2341 \bbbl@input@texini{##2}%
2342 \endgroup
2343 \else
2344 \xdef\bbbl@KVP@import{\bbbl@initoload}%
2345 \fi}%
2346 {}%
2347 \fi
2348 \ifx\bbbl@KVP@captions\@nil
2349 \let\bbbl@KVP@captions\bbbl@KVP@import
2350 \fi
2351 % ==
2352 \ifx\bbbl@KVP@transforms\@nil\else
2353 \bbbl@replace\bbbl@KVP@transforms{ },}%
2354 \fi
2355 % == Load ini ==
2356 \ifcase\bbbl@howloaded
2357 \bbbl@provide@new{##2}%
2358 \else
2359 \bbbl@ifblank{##1}%
2360 {}% With \bbbl@load@basic below
2361 {\bbbl@provide@renew{##2}}%
2362 \fi
2363 % Post tasks
2364 % -----
2365 % == subsequent calls after the first provide for a locale ==
2366 \ifx\bbbl@inidata\@empty\else
2367 \bbbl@extend@ini{##2}%
2368 \fi
2369 % == ensure captions ==
2370 \ifx\bbbl@KVP@captions\@nil\else
2371 \bbbl@ifunset{\bbbl@extracaps@##2}%
2372 {\bbbl@exp{\bbbl@babelensure[exclude=\\today]{##2}}}%
2373 {\bbbl@exp{\bbbl@babelensure[exclude=\\today,
2374 include=\bbbl@extracaps@##2]}{##2}}%
2375 \bbbl@ifunset{\bbbl@ensure@language}%
2376 {\bbbl@exp{%
2377 \\\DeclareRobustCommand\<\bbbl@ensure@language>[1]{%
2378 \\\foreignlanguage{language}%
2379 {###1}}}%
2380 {}%
2381 \bbbl@exp{%
2382 \\\bbbl@tglobal\<\bbbl@ensure@language>%
2383 \\\bbbl@tglobal\<\bbbl@ensure@language\space>%
2384 \fi
2385 % ==
2386 % At this point all parameters are defined if 'import'. Now we
2387 % execute some code depending on them. But what about if nothing was
2388 % imported? We just set the basic parameters, but still loading the
2389 % whole ini file.
2390 \bbbl@load@basic{##2}%
2391 % == script, language ==

```



```

2392 % Override the values from ini or defines them
2393 \ifx\bbl@KVP@script\@nil\else
2394   \bbl@csarg\edef\sname@#2{\bbl@KVP@script}%
2395 \fi
2396 \ifx\bbl@KVP@language\@nil\else
2397   \bbl@csarg\edef\lname@#2{\bbl@KVP@language}%
2398 \fi
2399 % == onchar ==
2400 \ifx\bbl@KVP@onchar\@nil\else
2401   \bbl@luahyphenate
2402   \directlua{
2403     if Babel.locale_mapped == nil then
2404       Babel.locale_mapped = true
2405       Babel.linebreaking.add_before(Babel.locale_map)
2406       Babel.loc_to_scr = {}
2407       Babel.chr_to_loc = Babel.chr_to_loc or {}
2408     end}%
2409   \bbl@xin@{ ids }{ \bbl@KVP@onchar\space}%
2410   \ifin@
2411     \ifx\bbl@starthyphens\@undefined % Needed if no explicit selection
2412       \AddBabelHook{babel-onchar}{beforestart}{\bbl@starthyphens}%
2413     \fi
2414     \bbl@exp{\bbl@add\bbl@starthyphens
2415       {\bbl@patterns@lua{\language}}}%
2416     % TODO - error/warning if no script
2417     \directlua{
2418       if Babel.script_blocks['\bbl@cl{sbc}'] then
2419         Babel.loc_to_scr[\the\localeid] =
2420           Babel.script_blocks['\bbl@cl{sbc}']
2421         Babel.locale_props[\the\localeid].lc = \the\localeid\space
2422         Babel.locale_props[\the\localeid].lg = \the\@nameuse{l@\language}\space
2423       end
2424     }%
2425   \fi
2426   \bbl@xin@{ fonts }{ \bbl@KVP@onchar\space}%
2427   \ifin@
2428     \bbl@ifunset{bbl@lsys@\language}{\bbl@provide@lsys{\language}}}%
2429     \bbl@ifunset{bbl@wdir@\language}{\bbl@provide@dirs{\language}}}%
2430     \directlua{
2431       if Babel.script_blocks['\bbl@cl{sbc}'] then
2432         Babel.loc_to_scr[\the\localeid] =
2433           Babel.script_blocks['\bbl@cl{sbc}']
2434       end}%
2435   \ifx\bbl@mapselect\@undefined % TODO. almost the same as mapfont
2436     \AtBeginDocument{%
2437       \bbl@patchfont{\bbl@mapselect}%
2438       {\selectfont}}%
2439     \def\bbl@mapselect{%
2440       \let\bbl@mapselect\relax
2441       \edef\bbl@prefontid{\fontid\font}}%
2442     \def\bbl@mapdir##1{%
2443       {\def\language{##1}%
2444         \let\bbl@ifrestoring\@firstoftwo % To avoid font warning
2445         \bbl@switchfont
2446         \directlua{
2447           Babel.locale_props[\the\csname bbl@id@##1\endcsname]
2448             [\the\bbl@prefontid] = \fontid\font\space}}}%
2449     \fi
2450     \bbl@exp{\bbl@add\bbl@mapselect{\bbl@mapdir{\language}}}%

```

```

2451 \fi
2452 % TODO - catch non-valid values
2453 \fi
2454 % == mapfont ==
2455 % For bidi texts, to switch the font based on direction
2456 \ifx\bbbl@KVP@mapfont\@nil\else
2457 \bbbl@ifsamestring{\bbbl@KVP@mapfont}{direction}}}%
2458 {\bbbl@error{Option '\bbbl@KVP@mapfont' unknown for\%
2459 mapfont. Use 'direction'.%
2460 {See the manual for details.}}}%
2461 \bbbl@ifunset{\bbbl@lsys@\languagename}{\bbbl@provide@lsys{\languagename}}}%
2462 \bbbl@ifunset{\bbbl@wdir@\languagename}{\bbbl@provide@dirs{\languagename}}}%
2463 \ifx\bbbl@mapselect\@undefined % TODO. See onchar.
2464 \AtBeginDocument{%
2465 \bbbl@patchfont{\bbbl@mapselect}}%
2466 {\selectfont}}%
2467 \def\bbbl@mapselect{%
2468 \let\bbbl@mapselect\relax
2469 \edef\bbbl@prefontid{\fontid\font}}%
2470 \def\bbbl@mapdir##1{%
2471 {\def\languagename{##1}%
2472 \let\bbbl@ifrestoring\@firstoftwo % avoid font warning
2473 \bbbl@switchfont
2474 \directlua{Babel.fontmap
2475 [\the\csname bbl@wdir@##1\endcsname]%
2476 [\bbbl@prefontid]=\fontid\font}}}%
2477 \fi
2478 \bbbl@exp{\bbbl@add\bbbl@mapselect{\bbbl@mapdir{\languagename}}}%
2479 \fi
2480 % == Line breaking: intraspace, intrapenalty ==
2481 % For CJK, East Asian, Southeast Asian, if interspace in ini
2482 \ifx\bbbl@KVP@intraspace\@nil\else % We can override the ini or set
2483 \bbbl@csarg\edef{intsp@#2}{\bbbl@KVP@intraspace}%
2484 \fi
2485 \bbbl@provide@intraspace
2486 % == Line breaking: CJK quotes ==
2487 \ifcase\bbbl@engine\or
2488 \bbbl@xin@{/c}{/\bbbl@c1{lnbrk}}}%
2489 \ifin@
2490 \bbbl@ifunset{\bbbl@quote@\languagename}}}%
2491 {\directlua{
2492 Babel.locale_props[\the\localeid].cjk_quotes = {}
2493 local cs = 'op'
2494 for c in string.utfvalues(%
2495 [[\csname bbl@quote@\languagename\endcsname]]) do
2496 if Babel.cjk_characters[c].c == 'qu' then
2497 Babel.locale_props[\the\localeid].cjk_quotes[c] = cs
2498 end
2499 cs = ( cs == 'op') and 'cl' or 'op'
2500 end
2501 }}%
2502 \fi
2503 \fi
2504 % == Line breaking: justification ==
2505 \ifx\bbbl@KVP@justification\@nil\else
2506 \let\bbbl@KVP@linebreaking\bbbl@KVP@justification
2507 \fi
2508 \ifx\bbbl@KVP@linebreaking\@nil\else
2509 \bbbl@xin@{,\bbbl@KVP@linebreaking,}{,elongated,kashida,cjk,unhyphenated,}%

```

```

2510 \ifin@
2511 \bbl@csarg\xdef
2512 {\lnbrk@\language\name}\xdef\car\bbl@KVP@linebreaking\@nil}%
2513 \fi
2514 \fi
2515 \bbl@xin@{/e}{/\bbl@cl{\lnbrk}}%
2516 \ifin\else\bbl@xin@{/k}{/\bbl@cl{\lnbrk}}\fi
2517 \ifin\bbl@arabicjust\fi
2518 % == Line breaking: hyphenate.other.(locale|script) ==
2519 \ifx\bbl@lbkflag\@empty
2520 \bbl@ifunset\bbl@hyotl@\language\name}%
2521 {\bbl@csarg\bbl@replace{hyotl@\language\name}{ }{,}%
2522 \bbl@startcommands*\language\name}%
2523 \bbl@csarg\bbl@foreach{hyotl@\language\name}%
2524 \ifcase\bbl@engine
2525 \ifnum##1<257
2526 \SetHyphenMap{\BabelLower{##1}{##1}}%
2527 \fi
2528 \else
2529 \SetHyphenMap{\BabelLower{##1}{##1}}%
2530 \fi}%
2531 \bbl@endcommands}%
2532 \bbl@ifunset\bbl@hyots@\language\name}%
2533 {\bbl@csarg\bbl@replace{hyots@\language\name}{ }{,}%
2534 \bbl@csarg\bbl@foreach{hyots@\language\name}%
2535 \ifcase\bbl@engine
2536 \ifnum##1<257
2537 \global\lccode##1=##1\relax
2538 \fi
2539 \else
2540 \global\lccode##1=##1\relax
2541 \fi}}%
2542 \fi
2543 % == Counters: maparabic ==
2544 % Native digits, if provided in ini (TeX level, xe and lua)
2545 \ifcase\bbl@engine\else
2546 \bbl@ifunset\bbl@dgnat@\language\name}%
2547 {\xdef\car\bbl@dgnat@\language\name\endcsname\@empty\else
2548 \xdef\car\bbl@dgnat@\language\name\endcsname
2549 \bbl@setdigits\csname\bbl@dgnat@\language\name\endcsname
2550 \ifx\bbl@KVP@maparabic\@nil\else
2551 \ifx\bbl@latinarabic\@undefined
2552 \xdef\car\bbl@latinarabic\@undefined
2553 \csname\bbl@counter@\language\name\endcsname
2554 \else % ie, if layout=counters, which redefines \@arabic
2555 \xdef\car\bbl@latinarabic
2556 \csname\bbl@counter@\language\name\endcsname
2557 \fi
2558 \fi
2559 \fi}%
2560 \fi
2561 % == Counters: mapdigits ==
2562 % Native digits (lua level).
2563 \ifodd\bbl@engine
2564 \ifx\bbl@KVP@mapdigits\@nil\else
2565 \bbl@ifunset\bbl@dgnat@\language\name}%
2566 {\RequirePackage{luatexbase}%
2567 \bbl@activate@preotf
2568 \directlua{

```

```

2569     Babel = Babel or {} %% -> presets in luababel
2570     Babel.digits_mapped = true
2571     Babel.digits = Babel.digits or {}
2572     Babel.digits[\the\localeid] =
2573       table.pack(string.utfvalue('\bbl@cl{dgnat}'))
2574     if not Babel.numbers then
2575       function Babel.numbers(head)
2576         local LOCALE = Babel.attr_locale
2577         local GLYPH = node.id'glyph'
2578         local inmath = false
2579         for item in node.traverse(head) do
2580           if not inmath and item.id == GLYPH then
2581             local temp = node.get_attribute(item, LOCALE)
2582             if Babel.digits[temp] then
2583               local chr = item.char
2584               if chr > 47 and chr < 58 then
2585                 item.char = Babel.digits[temp][chr-47]
2586               end
2587             end
2588             elseif item.id == node.id'math' then
2589               inmath = (item.subtype == 0)
2590             end
2591           end
2592         return head
2593       end
2594     end
2595   } }%
2596 \fi
2597 \fi
2598 % == Counters: alph, Alph ==
2599 % What if extras<lang> contains a \babel@save\@alph? It won't be
2600 % restored correctly when exiting the language, so we ignore
2601 % this change with the \bbl@alph@saved trick.
2602 \ifx\bbl@KVP@alph\@nil\else
2603   \bbl@extras@wrap{\bbl@alph@saved}%
2604   {\let\bbl@alph@saved\@alph}%
2605   {\let\@alph\bbl@alph@saved
2606     \babel@save\@alph}%
2607   \bbl@exp{%
2608     \bbl@add\<extras\language\>%
2609     \let\@alph<bbl@cntr@bbl@KVP@alph @\language\>}}%
2610 \fi
2611 \ifx\bbl@KVP@Alph\@nil\else
2612   \bbl@extras@wrap{\bbl@Alph@saved}%
2613   {\let\bbl@Alph@saved\@Alph}%
2614   {\let\@Alph\bbl@Alph@saved
2615     \babel@save\@Alph}%
2616   \bbl@exp{%
2617     \bbl@add\<extras\language\>%
2618     \let\@Alph<bbl@cntr@bbl@KVP@Alph @\language\>}}%
2619 \fi
2620 % == require.babel in ini ==
2621 % To load or reload the babel-*.tex, if require.babel in ini
2622 \ifx\bbl@beforestart\relax\else % But not in doc aux or body
2623   \bbl@ifunset{bbl@rqtex\@language\>}%
2624   {\expandafter\ifx\csname bbl@rqtex\@language\endcsname\@empty\else
2625     \let\BabelBeforeIni@gobbletwo
2626     \chardef\atcatcode=\catcode`\@
2627     \catcode`\@=11\relax

```

```

2628      \bbl@input@texini{\bbl@cs{rqtex@\language}}%
2629      \catcode`\@=\atcatcode
2630      \let\atcatcode\relax
2631      \global\bbl@csarg\let{rqtex@\language}\relax
2632      \fi}%
2633 \fi
2634 % == frenchspacing ==
2635 \ifcase\bbl@howloaded\in@true\else\in@false\fi
2636 \ifin@else\bbl@xin@{typography/frenchspacing}{\bbl@key@list}\fi
2637 \ifin@
2638   \bbl@extras@wrap{\bbl@pre@fs}%
2639   {\bbl@pre@fs}%
2640   {\bbl@post@fs}%
2641 \fi
2642 % == Release saved transforms ==
2643 \bbl@release@transforms\relax % \relax closes the last item.
2644 % == main ==
2645 \ifx\bbl@KVP@main\@nil % Restore only if not 'main'
2646   \let\language\bbl@savelangname
2647   \chardef\localeid\bbl@savelocaleid\relax
2648 \fi}

```

Depending on whether or not the language exists (based on \date<language>), we define two macros. Remember \bbl@startcommands opens a group.

```

2649 \def\bbl@provide@new#1{%
2650   \@namedef{date#1}{}% marks lang exists - required by \StartBabelCommands
2651   \@namedef{extras#1}{}%
2652   \@namedef{noextras#1}{}%
2653   \bbl@startcommands*{#1}{captions}%
2654   \ifx\bbl@KVP@captions\@nil % and also if import, implicit
2655     \def\bbl@tempb##1{% elt for \bbl@captionslist
2656       \ifx##1\@empty\else
2657         \bbl@exp{%
2658           \SetString\##1{%
2659             \bbl@nocaption{\bbl@stripslash##1}{#1\bbl@stripslash##1}}}%
2660         \expandafter\bbl@tempb
2661       \fi}%
2662     \expandafter\bbl@tempb\bbl@captionslist\@empty
2663   \else
2664     \ifx\bbl@initoload\relax
2665       \bbl@read@ini{\bbl@KVP@captions}2% % Here letters cat = 11
2666     \else
2667       \bbl@read@ini{\bbl@initoload}2% % Same
2668     \fi
2669   \fi
2670   \StartBabelCommands*{#1}{date}%
2671   \ifx\bbl@KVP@import\@nil
2672     \bbl@exp{%
2673       \SetString\today{\bbl@nocaption{today}{#1today}}}%
2674   \else
2675     \bbl@savetoday
2676     \bbl@savedate
2677   \fi
2678   \bbl@endcommands
2679   \bbl@load@basic{#1}%
2680   % == hyphenmins == (only if new)
2681   \bbl@exp{%
2682     \gdef\<#1hyphenmins>{%
2683       {\bbl@ifunset\bbl@ifthm@#1}{2}{\bbl@cs{lfthm@#1}}}%

```

```

2684     {\bbl@ifunset{\bbl@rgthm@#1}{3}{\bbl@cs{rgthm@#1}}}}}%
2685 % == hyphenrules (also in renew) ==
2686 \bbl@provide@hyphens{#1}%
2687 \ifx\bbl@KVP@main\@nil\else
2688   \expandafter\main@language\expandafter{#1}%
2689 \fi}
2690 %
2691 \def\bbl@provide@renew#1{%
2692   \ifx\bbl@KVP@captions\@nil\else
2693     \StartBabelCommands*{#1}{captions}%
2694     \bbl@read@ini{\bbl@KVP@captions}2%   % Here all letters cat = 11
2695     \EndBabelCommands
2696   \fi
2697   \ifx\bbl@KVP@import\@nil\else
2698     \StartBabelCommands*{#1}{date}%
2699     \bbl@savetoday
2700     \bbl@savestate
2701     \EndBabelCommands
2702   \fi
2703 % == hyphenrules (also in new) ==
2704 \ifx\bbl@lbkflag\@empty
2705   \bbl@provide@hyphens{#1}%
2706 \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. (TODO. But preserving previous values would be useful.)

```

2707 \def\bbl@load@basic#1{%
2708   \ifcase\bbl@howloaded\or\or
2709     \ifcase\csname bbl@llevel@\language\endcsname
2710       \bbl@csarg\let{lname@\language}\relax
2711     \fi
2712   \fi
2713   \bbl@ifunset{\bbl@lname@#1}%
2714   {\def\BabelBeforeIni##1##2{%
2715     \begingroup
2716       \let\bbl@ini@captions@aux\@gobbletwo
2717       \def\bbl@inidate #####1.####2.####3.####4\relax #####5####6}%
2718       \bbl@read@ini{##1}1%
2719       \ifx\bbl@initoload\relax\endinput\fi
2720     \endgroup}%
2721     \begingroup      % boxed, to avoid extra spaces:
2722     \ifx\bbl@initoload\relax
2723       \bbl@input@texini{##1}%
2724     \else
2725       \setbox\z@\hbox{\BabelBeforeIni{\bbl@initoload}}}%
2726     \fi
2727   \endgroup}%
2728   {}}

```

The hyphenrules option is handled with an auxiliary macro.

```

2729 \def\bbl@provide@hyphens#1{%
2730   \let\bbl@tempa\relax
2731   \ifx\bbl@KVP@hyphenrules\@nil\else
2732     \bbl@replace\bbl@KVP@hyphenrules{ }{,}%
2733     \bbl@foreach\bbl@KVP@hyphenrules{%
2734       \ifx\bbl@tempa\relax      % if not yet found
2735         \bbl@ifsamestring{##1}{+}%
2736         {\bbl@exp{\addlanguage\<1@##1>}}}%

```

```

2737     {}%
2738     \bbl@ifunset{l@##1}%
2739     {}%
2740     {\bbl@exp{\let\bbl@tempa<l@##1>}}%
2741     \fi}%
2742 \fi
2743 \ifx\bbl@tempa\relax %           if no opt or no language in opt found
2744     \ifx\bbl@KVP@import\@nil
2745         \ifx\bbl@initoload\relax\else
2746             \bbl@exp{%           and hyphenrules is not empty
2747                 \\bbl@ifblank{\bbl@cs{hyphr@#1}}%
2748                 {}%
2749                 {\let\\bbl@tempa<l@bbl@cl{hyphr}>}}%
2750             \fi
2751         \else % if importing
2752             \bbl@exp{%           and hyphenrules is not empty
2753                 \\bbl@ifblank{\bbl@cs{hyphr@#1}}%
2754                 {}%
2755                 {\let\\bbl@tempa<l@bbl@cl{hyphr}>}}%
2756             \fi
2757         \fi
2758     \bbl@ifunset{bbl@tempa}%     ie, relax or undefined
2759     {\bbl@ifunset{l@#1}%         no hyphenrules found - fallback
2760         {\bbl@exp{\\adddialect<l@#1>\language}}%
2761         {}}%                     so, l@<lang> is ok - nothing to do
2762     {\bbl@exp{\\adddialect<l@#1>\bbl@tempa}}% found in opt list or ini

```

The reader of babel-...tex files. We reset temporarily some catcodes.

```

2763 \def\bbl@input@texini#1{%
2764     \bbl@bsphack
2765     \bbl@exp{%
2766         \catcode`\\%=14 \catcode`\\=0
2767         \catcode`\\={1 \catcode`\\}=2
2768         \lowercase{\\InputIfFileExists{babel-#1.tex}{}}%
2769         \catcode`\\%=\the\catcode`\% \relax
2770         \catcode`\\=\the\catcode`\% \relax
2771         \catcode`\\={\the\catcode`\% \relax
2772         \catcode`\\=\the\catcode`\% \relax}%
2773     \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.

```

2774 \def\bbl@inline#1\bbl@inline{%
2775     \@ifnextchar[\bbl@iniset{\@ifnextchar\bbl@iniskip\bbl@inistore}#1\@@}% ]
2776 \def\bbl@iniset[#1]#2\@@{\def\bbl@section{#1}}
2777 \def\bbl@iniskip#1\@@{%         if starts with ;
2778 \def\bbl@inistore#1=#2\@@{%     full (default)
2779     \bbl@trim@def\bbl@tempa{#1}%
2780     \bbl@trim\toks@{#2}%
2781     \bbl@xin@{\bbl@section/\bbl@tempa}{\bbl@key@list}%
2782     \ifin@else
2783         \bbl@exp{%
2784             \\g@addto@macro\\bbl@inidata{%
2785                 \\bbl@elt{\bbl@section}{\bbl@tempa}{\the\toks@}}%
2786         \fi}
2787 \def\bbl@inistore@min#1=#2\@@{% minimal (maybe set in \bbl@read@ini)
2788     \bbl@trim@def\bbl@tempa{#1}%
2789     \bbl@trim\toks@{#2}%

```

```

2790 \bbl@xin@{.identification.}{.\bbl@section.}%
2791 \ifin@
2792 \bbl@exp{\g@addto@macro\bbl@inidata{%
2793 \bbl@elt{identification}{\bbl@tempa}{\the\toks@}}}%
2794 \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.

```

2795 \ifx\bbl@readstream\undefined
2796 \csname newread\endcsname\bbl@readstream
2797 \fi
2798 \def\bbl@read@ini#1#2{%
2799 \global\let\bbl@extend@ini\@gobble
2800 \openin\bbl@readstream=babel-#1.ini
2801 \ifeof\bbl@readstream
2802 \bbl@error
2803 {There is no ini file for the requested language\%
2804 (#1). Perhaps you misspelled it or your installation\%
2805 is not complete.}%
2806 {Fix the name or reinstall babel.}%
2807 \else
2808 % == Store ini data in \bbl@inidata ==
2809 \catcode`\[=12 \catcode`\]=12 \catcode`\==12 \catcode`\&=12
2810 \catcode`\;=12 \catcode`\|=12 \catcode`\%=14 \catcode`\-=12
2811 \bbl@info{Importing
2812 \ifcase#2font and identification \or basic \fi
2813 data for \language\%
2814 from babel-#1.ini. Reported}%
2815 \ifnum#2=\z@
2816 \global\let\bbl@inidata\@empty
2817 \let\bbl@inistore\bbl@inistore@min % Remember it's local
2818 \fi
2819 \def\bbl@section{identification}%
2820 \bbl@exp{\bbl@inistore tag.ini=#1\@@}%
2821 \bbl@inistore load.level=#2\@@
2822 \loop
2823 \if T\ifeof\bbl@readstream F\fi T\relax % Trick, because inside \loop
2824 \endlinechar\m@ne
2825 \read\bbl@readstream to \bbl@line
2826 \endlinechar\^^M
2827 \ifx\bbl@line\empty\else
2828 \expandafter\bbl@iniline\bbl@line\bbl@iniline
2829 \fi
2830 \repeat
2831 % == Process stored data ==
2832 \bbl@csarg\def{lini@language}{#1}%
2833 \bbl@read@ini@aux
2834 % == 'Export' data ==
2835 \bbl@ini@exports{#2}%
2836 \global\bbl@csarg\let{inidata@language}\bbl@inidata
2837 \global\let\bbl@inidata\@empty
2838 \bbl@exp{\bbl@add@list\bbl@ini@loaded{language}}%
2839 \bbl@tglobal\bbl@ini@loaded
2840 \fi}
2841 \def\bbl@read@ini@aux{%

```



```

2842 \let\bbl@savestrings\@empty
2843 \let\bbl@savetoday\@empty
2844 \let\bbl@savedate\@empty
2845 \def\bbl@elt##1##2##3{%
2846   \def\bbl@section{##1}%
2847   \in@{=date.}{=##1}% Find a better place
2848   \ifin@
2849     \bbl@ini@calendar{##1}%
2850   \fi
2851   \bbl@ifunset{bbl@inikv@##1}{}%
2852   {\csname bbl@inikv@##1\endcsname{##2}{##3}}}%
2853 \bbl@inidata}

```

A variant to be used when the ini file has been already loaded, because it's not the first \babelprovide for this language.

```

2854 \def\bbl@extend@ini@aux#1{%
2855   \bbl@startcommands*{#1}{captions}%
2856   % Activate captions/... and modify exports
2857   \bbl@csarg\def{inikv@captions.licr}##1##2{%
2858     \setlocalecaption{#1}{##1}{##2}%
2859   \def\bbl@inikv@captions##1##2{%
2860     \bbl@ini@captions@aux{##1}{##2}%
2861   \def\bbl@stringdef##1##2{\gdef##1{##2}}}%
2862   \def\bbl@exportkey##1##2##3{%
2863     \bbl@ifunset{bbl@kv@##2}{}%
2864     {\expandafter\ifx\csname bbl@kv@##2\endcsname\@empty\else
2865       \bbl@exp{\global\let<bbl@##1@<language>\<bbl@kv@##2>}%
2866       \fi}}%
2867   % As with \bbl@read@ini, but with some changes
2868   \bbl@read@ini@aux
2869   \bbl@ini@exports\tw@
2870   % Update inidata@lang by pretending the ini is read.
2871   \def\bbl@elt##1##2##3{%
2872     \def\bbl@section{##1}%
2873     \bbl@iniline##2=##3\bbl@iniline}%
2874     \csname bbl@inidata@#1\endcsname
2875     \global\bbl@csarg\let{inidata@#1}\bbl@inidata
2876   \StartBabelCommands*{#1}{date}% And from the import stuff
2877   \def\bbl@stringdef##1##2{\gdef##1{##2}}}%
2878   \bbl@savetoday
2879   \bbl@savedate
2880   \bbl@endcommands}

```

A somewhat hackish tool to handle calendar sections. To be improved.

```

2881 \def\bbl@ini@calendar#1{%
2882   \lowercase{\def\bbl@tempa{=##1=}}%
2883   \bbl@replace\bbl@tempa{=date.gregorian}{}%
2884   \bbl@replace\bbl@tempa{=date.}{}%
2885   \in@{.licr}{=##1=}%
2886   \ifin@
2887     \ifcase\bbl@engine
2888       \bbl@replace\bbl@tempa{.licr}{}%
2889     \else
2890       \let\bbl@tempa\relax
2891     \fi
2892   \fi
2893   \ifx\bbl@tempa\relax\else
2894     \bbl@replace\bbl@tempa{=}{}%
2895     \bbl@exp{%

```

```

2896 \def\<bbl@inikv@#1>####1####2{%
2897   \<bbl@inidate####1...\relax{####2}\<bbl@tempa}\}%
2898 \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).

```

2899 \def\bbl@renewinikey#1/#2\@#3{%
2900   \edef\bbl@tempa{\zap@space #1 \@empty}% section
2901   \edef\bbl@tempb{\zap@space #2 \@empty}% key
2902   \bbl@trim\toks@{#3}% value
2903   \bbl@exp{%
2904     \edef\\bbl@key@list{\bbl@key@list \bbl@tempa/\bbl@tempb;}%
2905     \\g@addto@macro\\bbl@inidata{%
2906       \\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.

```

2907 \def\bbl@exportkey#1#2#3{%
2908   \bbl@ifunset{\bbl@kv@#2}%
2909   {\bbl@csarg\gdef{#1\@language}{#3}}%
2910   {\xexpandafter\ifx\csname \bbl@kv@#2\endcsname\@empty
2911     \bbl@csarg\gdef{#1\@language}{#3}}%
2912   \else
2913     \bbl@exp{\global\let\<bbl@#1\@language>\<bbl@kv@#2>}%
2914   \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.

```

2915 \def\bbl@iniwarning#1{%
2916   \bbl@ifunset{\bbl@kv@identification.warning#1}{}%
2917   {\bbl@warning{%
2918     From babel-\bbl@cs{lini\@language}.ini:\\%
2919     \bbl@cs{@kv@identification.warning#1}\\%
2920     Reported }}}
2921 %
2922 \let\bbl@release@transforms\@empty
2923 %
2924 \def\bbl@ini@exports#1{%
2925   % Identification always exported
2926   \bbl@iniwarning{%
2927     \ifcase\bbl@engine
2928       \bbl@iniwarning{.pdflatex}%
2929     \or
2930       \bbl@iniwarning{.lualatex}%
2931     \or
2932       \bbl@iniwarning{.xelatex}%
2933     \fi%
2934     \bbl@exportkey{lllevel}{identification.load.level}{}%
2935     \bbl@exportkey{elname}{identification.name.english}{}%
2936     \bbl@exp{\\bbl@exportkey{lname}{identification.name.opentype}%
2937       {\csname \bbl@elname\@language\endcsname}}%
2938     \bbl@exportkey{tbcp}{identification.tag.bcp47}{}%
2939     \bbl@exportkey{lbcpl}{identification.language.tag.bcp47}{}%
2940     \bbl@exportkey{lotf}{identification.tag.opentype}{dflt}%
2941     \bbl@exportkey{esname}{identification.script.name}{}%
2942     \bbl@exp{\\bbl@exportkey{sname}{identification.script.name.opentype}%

```

```

2943   {\csname bbl@esname@language\endcsname}}%
2944   \bbl@exportkey{sbc}{identification.script.tag.bcp47}{}%
2945   \bbl@exportkey{sotf}{identification.script.tag.opentype}{DFLT}%
2946   % Also maps bcp47 -> language
2947   \ifbbl@bcptoname
2948     \bbl@csarg\xdef{bcp@map@bbl@cl{tbc}}{\language}%
2949   \fi
2950   % Conditional
2951   \ifnum#1>\z@           % 0 = only info, 1, 2 = basic, (re)new
2952     \bbl@exportkey{lnbrk}{typography.linebreaking}{h}%
2953     \bbl@exportkey{hyphr}{typography.hyphenrules}{}%
2954     \bbl@exportkey{lftm}{typography.lefthyphenmin}{2}%
2955     \bbl@exportkey{rgthm}{typography.righthyphenmin}{3}%
2956     \bbl@exportkey{prehc}{typography.prehyphenchar}{}%
2957     \bbl@exportkey{hyotl}{typography.hyphenate.other.locale}{}%
2958     \bbl@exportkey{hyots}{typography.hyphenate.other.script}{}%
2959     \bbl@exportkey{intsp}{typography.intraspaces}{u}%
2960     \bbl@exportkey{frspc}{typography.frenchspacing}{u}%
2961     \bbl@exportkey{chrng}{characters.ranges}{}%
2962     \bbl@exportkey{quote}{characters.delimiters.quotes}{}%
2963     \bbl@exportkey{dgnat}{numbers.digits.native}{}%
2964     \ifnum#1=\tw@       % only (re)new
2965       \bbl@exportkey{rqtex}{identification.require.babel}{}%
2966       \bbl@toglobal\bbl@savetoday
2967       \bbl@toglobal\bbl@savestate
2968       \bbl@savestrings
2969     \fi
2970   \fi}

```

A shared handler for key=val lines to be stored in \bbl@kv@<section>.<key>.

```

2971 \def\bbl@inikv#1#2{%      key=value
2972   \toks@{#2}%             This hides #'s from ini values
2973   \bbl@csarg\xdef{kv@bbl@section.#1}{\the\toks@}}

```

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

```

2974 \let\bbl@inikv@identification\bbl@inikv
2975 \let\bbl@inikv@typography\bbl@inikv
2976 \let\bbl@inikv@characters\bbl@inikv
2977 \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’.

```

2978 \def\bbl@inikv@counters#1#2{%
2979   \bbl@ifsamestring{#1}{digits}%
2980   {\bbl@error{The counter name 'digits' is reserved for mapping\\
2981     decimal digits}%
2982     {Use another name.}}%
2983   }%
2984   \def\bbl@tempc{#1}%
2985   \bbl@trim@def{\bbl@tempb*}{#2}%
2986   \in@{.1$}{#1$}%
2987   \ifin@
2988     \bbl@replace\bbl@tempc{.1}{}%
2989     \bbl@csarg\protected@xdef{cntr@bbl@tempc @language}{%
2990       \noexpand\bbl@alphanumeric{\bbl@tempc}}%
2991   \fi
2992   \in@{.F.}{#1}%
2993   \ifin@else\in@{.S.}{#1}\fi

```

```

2994 \ifin@
2995 \bbl@csarg\protected@xdef{cnt@#1@\language}\bbl@tempb*}%
2996 \else
2997 \toks@{}% Required by \bbl@buildifcase, which returns \bbl@tempa
2998 \expandafter\bbl@buildifcase\bbl@tempb* \ \ % Space after \
2999 \bbl@csarg{\global\expandafter\let}{cnt@#1@\language}\bbl@tempa
3000 \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.

```

3001 \ifcase\bbl@engine
3002 \bbl@csarg\def{inikv@captions.licr}#1#2{%
3003 \bbl@ini@captions@aux{#1}{#2}}
3004 \else
3005 \def\bbl@inikv@captions#1#2{%
3006 \bbl@ini@captions@aux{#1}{#2}}
3007 \fi

```

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

```

3008 \def\bbl@ini@captions@template#1#2{% string language tempa=capt-name
3009 \bbl@replace\bbl@tempa{.template}{}%
3010 \def\bbl@toreplace{#1}{}%
3011 \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace}}%
3012 \bbl@replace\bbl@toreplace{[ ]}{\csname}%
3013 \bbl@replace\bbl@toreplace{[ ]}{\csname the}%
3014 \bbl@replace\bbl@toreplace{[ ]}{name\endcsname}}%
3015 \bbl@replace\bbl@toreplace{[ ]}{\endcsname}}%
3016 \bbl@xin@{, \bbl@tempa,}{, chapter, appendix, part,}%
3017 \ifin@
3018 \@nameuse{bbl@patch\bbl@tempa}%
3019 \global\bbl@csarg\let{\bbl@tempa fmt@#2}\bbl@toreplace
3020 \fi
3021 \bbl@xin@{, \bbl@tempa,}{, figure, table,}%
3022 \ifin@
3023 \toks@\expandafter{\bbl@toreplace}%
3024 \bbl@exp{\gdef\<fnun@\bbl@tempa>{\the\toks@}}%
3025 \fi}
3026 \def\bbl@ini@captions@aux#1#2{%
3027 \bbl@trim@def\bbl@tempa{#1}%
3028 \bbl@xin@{.template}{\bbl@tempa}%
3029 \ifin@
3030 \bbl@ini@captions@template{#2}\language
3031 \else
3032 \bbl@ifblank{#2}%
3033 {\bbl@exp{%
3034 \toks@{\bbl@nocaption{\bbl@tempa}{\language\bbl@tempa name}}}%
3035 {\bbl@trim\toks@{#2}}}%
3036 \bbl@exp{%
3037 \bbl@add{\bbl@savestrings{%
3038 \SetString\<\bbl@tempa name>{\the\toks@}}}%
3039 \toks@\expandafter{\bbl@captionslist}%
3040 \bbl@exp{\in{\<\bbl@tempa name>}{\the\toks@}}}%
3041 \ifin@else
3042 \bbl@exp{%
3043 \bbl@add{\bbl@extracaps@\language}{\<\bbl@tempa name>}%
3044 \bbl@tglobal{\bbl@extracaps@\language}%
3045 \fi
3046 \fi}

```

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

```

3047 \def\bbl@list@the{%
3048   part,chapter,section,subsection,subsubsection,paragraph,%
3049   subparagraph,enumi,enumii,enumiii,enumiv,equation,figure,%
3050   table,page,footnote,mpfootnote,mpfn}
3051 \def\bbl@map@cnt#1{% #1:roman,etc, // #2:enumi,etc
3052   \bbl@ifunset{bbl@map@#1@\languagename}%
3053     {\@nameuse{#1}}%
3054     {\@nameuse{bbl@map@#1@\languagename}}}%
3055 \def\bbl@inikv@labels#1#2{%
3056   \in@{.map}{#1}%
3057   \ifin@
3058     \ifx\bbl@KVP@labels\@nil\else
3059       \bbl@xin@{ map }{ \bbl@KVP@labels\space}%
3060       \ifin@
3061         \def\bbl@tempc{#1}%
3062         \bbl@replace\bbl@tempc{.map}{}%
3063         \in@{,#2,}{,arabic,roman,Roman,alph,Alph,fnsymbol,}%
3064         \bbl@exp{%
3065           \gdef\<bbl@map@\bbl@tempc @\languagename>%
3066             {\ifin@\<#2>\else\\localecounter{#2}\fi}}%
3067         \bbl@foreach\bbl@list@the{%
3068           \bbl@ifunset{the##1}{}%
3069             {\bbl@exp{\let\\bbl@tempd\<the##1>}%
3070               \bbl@exp{%
3071                 \\bbl@sreplace\<the##1>%
3072                 {\<\bbl@tempc>{##1}}{\\\bbl@map@cnt{\bbl@tempc}{##1}}}%
3073                 \\bbl@sreplace\<the##1>%
3074                 {\<\empty @\bbl@tempc>\<c@##1>}{\\bbl@map@cnt{\bbl@tempc}{##1}}}%
3075               \expandafter\ifx\csname the##1\endcsname\bbl@tempd\else
3076                 \toks@\expandafter\expandafter\expandafter{%
3077                   \csname the##1\endcsname}%
3078                 \expandafter\xdef\csname the##1\endcsname{{\the\toks@}}%
3079                 \fi}}%
3080       \fi
3081     \fi
3082   %
3083   \else
3084     %
3085     % The following code is still under study. You can test it and make
3086     % suggestions. Eg, enumerate.2 = ([enumi]).([enumii]). It's
3087     % language dependent.
3088     \in@{enumerate.}{#1}%
3089     \ifin@
3090       \def\bbl@tempa{#1}%
3091       \bbl@replace\bbl@tempa{enumerate.}{}%
3092       \def\bbl@toreplace{#2}%
3093       \bbl@replace\bbl@toreplace{[ ]}{\nobreakspace{}}%
3094       \bbl@replace\bbl@toreplace{[ ]}{\csname the}%
3095       \bbl@replace\bbl@toreplace{ ]}{\endcsname{}}}%
3096       \toks@\expandafter{\bbl@toreplace}%
3097       % TODO. Execute only once:
3098       \bbl@exp{%
3099         \\bbl@add\<extras\languagename>%
3100         \\babel@save\<labelenum\romannumeral\bbl@tempa>%
3101         \def\<labelenum\romannumeral\bbl@tempa>{\the\toks@}}%
3102         \\bbl@toggle\<extras\languagename>%
3103     \fi

```

3104 \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.

```

3105 \def\bbl@chapttype{chapter}
3106 \ifx\@makechapterhead\@undefined
3107   \let\bbl@patchchapter\relax
3108 \else\ifx\thechapter\@undefined
3109   \let\bbl@patchchapter\relax
3110 \else\ifx\ps@headings\@undefined
3111   \let\bbl@patchchapter\relax
3112 \else
3113   \def\bbl@patchchapter{%
3114     \global\let\bbl@patchchapter\relax
3115     \gdef\bbl@chfmt{%
3116       \bbl@ifunset{bbl@\bbl@chapttype fmt@\language\name}%
3117       {\@chapapp\space\thechapter}
3118       {\@nameuse{bbl@\bbl@chapttype fmt@\language\name}}}%
3119     \bbl@add\appendix{\def\bbl@chapttype{appendix}}% Not harmful, I hope
3120     \bbl@sreplace\ps@headings{\@chapapp\ \thechapter}{\bbl@chfmt}%
3121     \bbl@sreplace\chaptermark{\@chapapp\ \thechapter}{\bbl@chfmt}%
3122     \bbl@sreplace\@makechapterhead{\@chapapp\space\thechapter}{\bbl@chfmt}%
3123     \bbl@tglobal\appendix
3124     \bbl@tglobal\ps@headings
3125     \bbl@tglobal\chaptermark
3126     \bbl@tglobal\@makechapterhead}
3127   \let\bbl@patchappendix\bbl@patchchapter
3128 \fi\fi\fi
3129 \ifx\@part\@undefined
3130   \let\bbl@patchpart\relax
3131 \else
3132   \def\bbl@patchpart{%
3133     \global\let\bbl@patchpart\relax
3134     \gdef\bbl@partformat{%
3135       \bbl@ifunset{bbl@partfmt@\language\name}%
3136       {\partname\nobreakspace\thepart}
3137       {\@nameuse{bbl@partfmt@\language\name}}}%
3138     \bbl@sreplace\@part{\partname\nobreakspace\thepart}{\bbl@partformat}%
3139     \bbl@tglobal\@part}
3140 \fi

```

**Date.** TODO. Document

```

3141 % Arguments are _not_ protected.
3142 \let\bbl@calendar\@empty
3143 \DeclareRobustCommand\localedate[1][\bbl@localedate{#1}]{}
3144 \def\bbl@localedate#1#2#3#4{%
3145   \begingroup
3146     \ifx\@empty#1\@empty\else
3147       \let\bbl@ld@calendar\@empty
3148       \let\bbl@ld@variant\@empty
3149       \edef\bbl@tempa{\zap@space#1 \@empty}%
3150       \def\bbl@tempb##1=##2\@{\@namedef{bbl@ld##1}{##2}}%
3151       \bbl@foreach\bbl@tempa{\bbl@tempb##1\@}%
3152       \edef\bbl@calendar{%
3153         \bbl@ld@calendar
3154         \ifx\bbl@ld@variant\@empty\else
3155           .\bbl@ld@variant

```

```

3156         \fi}%
3157         \bbl@replace\bbl@calendar{gregorian}{}%
3158         \fi
3159         \bbl@cased
3160         {\@nameuse{bbl@date@\languagename @\bbl@calendar}{#2}{#3}{#4}}%
3161     \endgroup}
3162 % eg: 1=months, 2=wide, 3=1, 4=dummy, 5=value, 6=calendar
3163 \def\bbl@inidate#1.#2.#3.#4\relax#5#6{% TODO - ignore with 'captions'
3164     \bbl@trim@def\bbl@tempa{#1.#2}%
3165     \bbl@ifsamestring{\bbl@tempa}{months.wide}%         to savedate
3166     {\bbl@trim@def\bbl@tempa{#3}%
3167         \bbl@trim\toks@{#5}%
3168         \@temptokena\expandafter{\bbl@savestate}%
3169         \bbl@exp{% Reverse order - in ini last wins
3170             \def\\bbl@savestate{%
3171                 \\SetString\<month\romannumeral\bbl@tempa#6name>{\the\toks@}%
3172                 \the\@temptokena}}}%
3173     {\bbl@ifsamestring{\bbl@tempa}{date.long}%         defined now
3174         {\lowercase{\def\bbl@tempb{#6}}%
3175             \bbl@trim@def\bbl@toreplace{#5}%
3176             \bbl@TG@@date
3177             \bbl@ifunset{bbl@date@\languagename @}%
3178             {\bbl@exp{% TODO. Move to a better place.
3179                 \gdef\<\languagename date>{\protect\<\languagename date >}%
3180                 \gdef\<\languagename date >####1####2####3{%
3181                     \\bbl@usedategroupttrue
3182                     \<bbl@ensure@\languagename>{%
3183                         \\localedate{####1}{####2}{####3}}}%
3184                     \\bbl@add\\bbl@savetoday{%
3185                         \\SetString\\today{%
3186                             \<\languagename date>%
3187                             {\the\year}{\the\month}{\the\day}}}%
3188                     }%
3189                     \global\bbl@csarg\let{date@\languagename @}\bbl@toreplace
3190                     \ifx\bbl@tempb\@empty\else
3191                         \global\bbl@csarg\let{date@\languagename @\bbl@tempb}\bbl@toreplace
3192                     \fi}%
3193                 }}}

```

**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. Note after `\bbl@replace \toks@` contains the resulting string, which is used by `\bbl@replace@finish@iii` (this implicit behavior doesn’t seem a good idea, but it’s efficient).

```

3194 \let\bbl@calendar\@empty
3195 \newcommand\BabelDateSpace{\nobreakspace}
3196 \newcommand\BabelDateDot{.\@} % TODO. \let instead of repeating
3197 \newcommand\BabelDated[1]{\number#1}
3198 \newcommand\BabelDatedd[1]{\ifnum#1<10 0\fi\number#1}
3199 \newcommand\BabelDateM[1]{\number#1}
3200 \newcommand\BabelDateMM[1]{\ifnum#1<10 0\fi\number#1}
3201 \newcommand\BabelDateMMM[1]{%
3202     \csname month\romannumeral#1\bbl@calendar name\endcsname}%
3203 \newcommand\BabelDatey[1]{\number#1}%
3204 \newcommand\BabelDateyy[1]{%
3205     \ifnum#1<10 0\number#1 %
3206     \else\ifnum#1<100 \number#1 %
3207     \else\ifnum#1<1000 \expandafter\@gobble\number#1 %
3208     \else\ifnum#1<10000 \expandafter\@gobbletwo\number#1 %

```

```

3209 \else
3210 \bbl@error
3211 {Currently two-digit years are restricted to the\
3212 range 0-9999.}%
3213 {There is little you can do. Sorry.}%
3214 \fi\fi\fi\fi}
3215 \newcommand\BabelDateyyyy[1]{\number#1} % TODO - add leading 0
3216 \def\bbl@replace@finish@iii#1{%
3217 \bbl@exp{\def\#1###1###2###3{\the\toks@}}
3218 \def\bbl@TG@date{%
3219 \bbl@replace\bbl@toreplace{[ ]}{\BabelDateSpace{}}%
3220 \bbl@replace\bbl@toreplace{[.]}{\BabelDateDot{}}%
3221 \bbl@replace\bbl@toreplace{[d]}{\BabelDated{###3}}%
3222 \bbl@replace\bbl@toreplace{[dd]}{\BabelDatedd{###3}}%
3223 \bbl@replace\bbl@toreplace{[M]}{\BabelDateM{###2}}%
3224 \bbl@replace\bbl@toreplace{[MM]}{\BabelDateMM{###2}}%
3225 \bbl@replace\bbl@toreplace{[MMM]}{\BabelDateMMM{###2}}%
3226 \bbl@replace\bbl@toreplace{[y]}{\BabelDatey{###1}}%
3227 \bbl@replace\bbl@toreplace{[yy]}{\BabelDateyy{###1}}%
3228 \bbl@replace\bbl@toreplace{[yyyy]}{\BabelDateyyyy{###1}}%
3229 \bbl@replace\bbl@toreplace{[y|]}{\bbl@datecctr[###1|]}%
3230 \bbl@replace\bbl@toreplace{[m|]}{\bbl@datecctr[###2|]}%
3231 \bbl@replace\bbl@toreplace{[d|]}{\bbl@datecctr[###3|]}%
3232 \bbl@replace@finish@iii\bbl@toreplace}
3233 \def\bbl@datecctr{\expandafter\bbl@xdatecctr\expandafter}
3234 \def\bbl@xdatecctr[#1|#2]{\localenumeral{#2}{#1}}

```

#### Transforms.

```

3235 \let\bbl@release@transforms\@empty
3236 \@namedef{bbl@inikv@transforms.prehyphenation}{%
3237 \bbl@transforms\babelprehyphenation}
3238 \@namedef{bbl@inikv@transforms.posthyphenation}{%
3239 \bbl@transforms\babelposthyphenation}
3240 \def\bbl@transforms@aux#1#2#3,#4\relax{#1{#2}{#3}{#4}}
3241 \begingroup % A hack. TODO. Don't require an specific order
3242 \catcode\%=12
3243 \catcode\&=14
3244 \gdef\bbl@transforms#1#2#3{&%
3245 \ifx\bbl@KVP@transforms\@nil\else
3246 \directlua{
3247 str = [==[#2]==]
3248 str = str:gsub('%.%d+%.%d+$', '')
3249 tex.print([[ \def\string\babeltempa{]] .. str .. [ ]]])
3250 }&%
3251 \bbl@xin@{,\babeltempa,}{,\bbl@KVP@transforms,}&%
3252 \ifin@
3253 \in@{.0$}{#2$}&%
3254 \ifin@
3255 \g@addto@macro\bbl@release@transforms{&%
3256 \relax\bbl@transforms@aux#1{\languagenname}{#3}}&%
3257 \else
3258 \g@addto@macro\bbl@release@transforms{, {#3}}&%
3259 \fi
3260 \fi
3261 \fi}
3262 \endgroup

```

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



```

3263 \def\bbl@provide@lsys#1{%
3264   \bbl@ifunset{bbl@lname@#1}%
3265     {\bbl@load@info{#1}}%
3266     }%
3267   \bbl@csarg\let{lsys@#1}\@empty
3268   \bbl@ifunset{bbl@sname@#1}{\bbl@csarg\gdef{sname@#1}{Default}}{}%
3269   \bbl@ifunset{bbl@sotf@#1}{\bbl@csarg\gdef{sotf@#1}{DFLT}}{}%
3270   \bbl@csarg\bbl@add@list{lsys@#1}{Script=\bbl@cs{sname@#1}}%
3271   \bbl@ifunset{bbl@lname@#1}{}%
3272     {\bbl@csarg\bbl@add@list{lsys@#1}{Language=\bbl@cs{lname@#1}}}%
3273   \ifcase\bbl@engine\or\or
3274     \bbl@ifunset{bbl@prehc@#1}{}%
3275       {\bbl@exp{\bbl@ifblank{\bbl@cs{prehc@#1}}}%
3276         }%
3277       {\ifx\bbl@xenoxyph\@undefined
3278         \let\bbl@xenoxyph\bbl@xenoxyph@d
3279         \ifx\AtBeginDocument\@notprerr
3280           \expandafter\@secondoftwo % to execute right now
3281           \fi
3282         \AtBeginDocument{%
3283           \bbl@patchfont{\bbl@xenoxyph}%
3284           \expandafter\selectlanguage\expandafter{\language}%
3285         \fi}%
3286   \fi
3287   \bbl@csarg\bbl@tglobal{lsys@#1}}
3288 \def\bbl@xenoxyph@d{%
3289   \bbl@ifset{bbl@prehc@language}%
3290     {\ifnum\hyphenchar\font=\defaultshyphenchar
3291       \iffontchar\font\bbl@c1{prehc}\relax
3292       \hyphenchar\font\bbl@c1{prehc}\relax
3293       \else\iffontchar\font"200B
3294       \hyphenchar\font"200B
3295       \else
3296         \bbl@warning
3297           {Neither 0 nor ZERO WIDTH SPACE are available\\%
3298            in the current font, and therefore the hyphen\\%
3299            will be printed. Try changing the fontspec's\\%
3300            'HyphenChar' to another value, but be aware\\%
3301            this setting is not safe (see the manual)}%
3302         \hyphenchar\font\defaultshyphenchar
3303       \fi\fi
3304     \fi}%
3305   {\hyphenchar\font\defaultshyphenchar}}
3306 % \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).

```

3307 \def\bbl@load@info#1{%
3308   \def\BabelBeforeIni##1##2{%
3309     \begingroup
3310       \bbl@read@ini{##1}0%
3311       \endinput          % babel- .tex may contain onlypreamble's
3312       \endgroup}%        boxed, to avoid extra spaces:
3313   {\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]

```

3345 \def\bbl@buildifcase#1 {% Returns \bbl@tempa, requires \toks@={%
3346   \ifx\\#1%           % \\ before, in case #1 is multiletter
3347     \bbl@exp{%
3348       \def\\ \bbl@tempa####1{%
3349         \<ifcase>####1\space\the\toks@\<else>\\ \ctrerr\<fi>}}%
3350   \else
3351     \toks@\expandafter{\the\toks@\or #1}%
3352     \expandafter\bbl@buildifcase
3353 \fi}

```

```

3354 \newcommand\localexnumeral[2]{\bbl@cs{cntr@#1\@language}{#2}}
3355 \def\bbl@localexcntr#1#2{\localexnumeral{#2}{#1}}
3356 \newcommand\localecounter[2]{%
3357   \expandafter\bbl@localexcntr
3358   \expandafter{\number\csname c@#2\endcsname}{#1}}
3359 \def\bbl@alphnumeral#1#2{%
3360   \expandafter\bbl@alphnumeral@i\number#2 76543210\@@{#1}}
3361 \def\bbl@alphnumeral@i#1#2#3#4#5#6#7#8\@@#9{%
3362   \ifcase\car#8\@nilor % Currently <10000, but prepared for bigger
3363     \bbl@alphnumeral@ii{#9}000000#1\or

```

```

3364 \bbl@alphanumeric@ii{#9}0000#1#2\or
3365 \bbl@alphanumeric@ii{#9}0000#1#2#3\or
3366 \bbl@alphanumeric@ii{#9}000#1#2#3#4\else
3367 \bbl@alphnum@invalid{>9999}%
3368 \fi}
3369 \def\bbl@alphanumeric@ii#1#2#3#4#5#6#7#8{%
3370 \bbl@ifunset{bbl@cntr@#1.F.\number#5#6#7#8@\language}%
3371 {\bbl@cs{cntr@#1.4@\language}#5%
3372 \bbl@cs{cntr@#1.3@\language}#6%
3373 \bbl@cs{cntr@#1.2@\language}#7%
3374 \bbl@cs{cntr@#1.1@\language}#8%
3375 \ifnum#6#7#8>\z@ % TODO. An ad hoc rule for Greek. Ugly.
3376 \bbl@ifunset{bbl@cntr@#1.S.321@\language}{}%
3377 {\bbl@cs{cntr@#1.S.321@\language}}%
3378 \fi}%
3379 {\bbl@cs{cntr@#1.F.\number#5#6#7#8@\language}}%
3380 \def\bbl@alphnum@invalid#1{%
3381 \bbl@error{Alphabetic numeral too large (#1)}%
3382 {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.

```

3383 \newcommand\localeinfo[1]{%
3384 \bbl@ifunset{bbl@csname bbl@info@#1\endcsname @\language}%
3385 {\bbl@error{I've found no info for the current locale.\%
3386 The corresponding ini file has not been loaded\%
3387 Perhaps it doesn't exist}%
3388 {See the manual for details.}}%
3389 {\bbl@cs{csname bbl@info@#1\endcsname @\language}}%
3390 % \@namedef{bbl@info@name.locale}{lname}
3391 \@namedef{bbl@info@tag.ini}{lini}
3392 \@namedef{bbl@info@name.english}{elname}
3393 \@namedef{bbl@info@name.opentype}{lname}
3394 \@namedef{bbl@info@tag.bcp47}{tbc}
3395 \@namedef{bbl@info@language.tag.bcp47}{lbcp}
3396 \@namedef{bbl@info@tag.opentype}{lotf}
3397 \@namedef{bbl@info@script.name}{esname}
3398 \@namedef{bbl@info@script.name.opentype}{sname}
3399 \@namedef{bbl@info@script.tag.bcp47}{sbcp}
3400 \@namedef{bbl@info@script.tag.opentype}{sotf}
3401 \let\bbl@ensureinfo\@gobble
3402 \newcommand\BabelEnsureInfo{%
3403 \ifx\InputIfFileExists\undefined\else
3404 \def\bbl@ensureinfo##1{%
3405 \bbl@ifunset{bbl@lname@##1}{\bbl@load@info{##1}}}%
3406 \fi
3407 \bbl@foreach\bbl@loaded{%
3408 \def\language{##1}%
3409 \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`.

```

3410 \newcommand\getlocaleproperty{%
3411 \@ifstar\bbl@getproperty@s\bbl@getproperty@x}
3412 \def\bbl@getproperty@s#1#2#3{%
3413 \let#1\relax
3414 \def\bbl@elt##1##2##3{%
3415 \bbl@ifsamestring{##1/##2}{#3}%

```

```

3416      {\providecommand#1{##3}%
3417      \def\bbl@elt####1####2####3{}}}%
3418      {}}}%
3419      \bbl@cs{inidata@#2}}}%
3420 \def\bbl@getproperty@x#1#2#3{%
3421   \bbl@getproperty@s{#1}{#2}{#3}%
3422   \ifx#1\relax
3423     \bbl@error
3424     {Unknown key for locale '#2':\%
3425     #3\}%
3426     \string#1 will be set to \relax}%
3427     {Perhaps you misspelled it.}%
3428   \fi}
3429 \let\bbl@ini@loaded\@empty
3430 \newcommand\LocaleForEach{\bbl@foreach\bbl@ini@loaded}

```

## 9 Adjusting the Babel behavior

A generic high level interface is provided to adjust some global and general settings.

```

3431 \newcommand\babeladjust[1]{% TODO. Error handling.
3432   \bbl@forkv{#1}{%
3433     \bbl@ifunset{\bbl@ADJ@##1@##2}%
3434     {\bbl@cs{ADJ@##1}{##2}}}%
3435     {\bbl@cs{ADJ@##1@##2}}}
3436 %
3437 \def\bbl@adjust@lua#1#2{%
3438   \ifvmode
3439     \ifnum\currentgrouplevel=\z@
3440       \directlua{ Babel.#2 }%
3441       \expandafter\expandafter\expandafter\@gobble
3442     \fi
3443   \fi
3444   {\bbl@error % The error is gobbled if everything went ok.
3445     {Currently, #1 related features can be adjusted only\%
3446     in the main vertical list.}%
3447     {Maybe things change in the future, but this is what it is.}}}
3448 \@namedef{\bbl@ADJ@bidi.mirroring@on}{%
3449   \bbl@adjust@lua{bidi}{mirroring_enabled=true}}
3450 \@namedef{\bbl@ADJ@bidi.mirroring@off}{%
3451   \bbl@adjust@lua{bidi}{mirroring_enabled=false}}
3452 \@namedef{\bbl@ADJ@bidi.text@on}{%
3453   \bbl@adjust@lua{bidi}{bidi_enabled=true}}
3454 \@namedef{\bbl@ADJ@bidi.text@off}{%
3455   \bbl@adjust@lua{bidi}{bidi_enabled=false}}
3456 \@namedef{\bbl@ADJ@bidi.mapdigits@on}{%
3457   \bbl@adjust@lua{bidi}{digits_mapped=true}}
3458 \@namedef{\bbl@ADJ@bidi.mapdigits@off}{%
3459   \bbl@adjust@lua{bidi}{digits_mapped=false}}
3460 %
3461 \@namedef{\bbl@ADJ@linebreak.sea@on}{%
3462   \bbl@adjust@lua{linebreak}{sea_enabled=true}}
3463 \@namedef{\bbl@ADJ@linebreak.sea@off}{%
3464   \bbl@adjust@lua{linebreak}{sea_enabled=false}}
3465 \@namedef{\bbl@ADJ@linebreak.cjk@on}{%
3466   \bbl@adjust@lua{linebreak}{cjk_enabled=true}}
3467 \@namedef{\bbl@ADJ@linebreak.cjk@off}{%
3468   \bbl@adjust@lua{linebreak}{cjk_enabled=false}}
3469 \@namedef{\bbl@ADJ@justify.arabic@on}{%

```

```

3470 \bbl@adjust@lua{linebreak}{arabic.justify_enabled=true}}
3471 \@namedef{bbl@ADJ@justify.arabic@off}{%
3472 \bbl@adjust@lua{linebreak}{arabic.justify_enabled=false}}
3473 %
3474 \def\bbl@adjust@layout#1{%
3475 \ifvmode
3476 #1%
3477 \expandafter\@gobble
3478 \fi
3479 {\bbl@error % The error is gobbled if everything went ok.
3480 {Currently, layout related features can be adjusted only\%
3481 in vertical mode.}%
3482 {Maybe things change in the future, but this is what it is.}}}
3483 \@namedef{bbl@ADJ@layout.tabular@on}{%
3484 \bbl@adjust@layout{\let\@tabular\bbl@NL@@tabular}}
3485 \@namedef{bbl@ADJ@layout.tabular@off}{%
3486 \bbl@adjust@layout{\let\@tabular\bbl@OL@@tabular}}
3487 \@namedef{bbl@ADJ@layout.lists@on}{%
3488 \bbl@adjust@layout{\let\list\bbl@NL@list}}
3489 \@namedef{bbl@ADJ@layout.lists@off}{%
3490 \bbl@adjust@layout{\let\list\bbl@OL@list}}
3491 \@namedef{bbl@ADJ@hyphenation.extra@on}{%
3492 \bbl@activateposthyphen}
3493 %
3494 \@namedef{bbl@ADJ@autoload.bcp47@on}{%
3495 \bbl@bcpallowedtrue}
3496 \@namedef{bbl@ADJ@autoload.bcp47@off}{%
3497 \bbl@bcpallowedfalse}
3498 \@namedef{bbl@ADJ@autoload.bcp47.prefix}#1{%
3499 \def\bbl@bcp@prefix{#1}}
3500 \def\bbl@bcp@prefix{bcp47-}
3501 \@namedef{bbl@ADJ@autoload.options}#1{%
3502 \def\bbl@autoload@options{#1}}
3503 \let\bbl@autoload@bcptoptions\@empty
3504 \@namedef{bbl@ADJ@autoload.bcp47.options}#1{%
3505 \def\bbl@autoload@bcptoptions{#1}}
3506 \newif\ifbbl@bcptname
3507 \@namedef{bbl@ADJ@bcp47.toname@on}{%
3508 \bbl@bcptnametrue}
3509 \BabelEnsureInfo}
3510 \@namedef{bbl@ADJ@bcp47.toname@off}{%
3511 \bbl@bcptnamefalse}
3512 \@namedef{bbl@ADJ@prehyphenation.disable@nohyphenation}{%
3513 \directlua{ Babel.ignore_pre_char = function(node)
3514 return (node.lang == \the\csname l@nohyphenation\endcsname)
3515 end }}
3516 \@namedef{bbl@ADJ@prehyphenation.disable@off}{%
3517 \directlua{ Babel.ignore_pre_char = function(node)
3518 return false
3519 end }}

```

As the final task, load the code for lua. TODO: use babel name, override

```

3520 \ifx\directlua\@undefined\else
3521 \ifx\bbl@luapatterns\@undefined
3522 \input luababel.def
3523 \fi
3524 \fi

```

Continue with  $\LaTeX$ .

```

3525 </package | core>
3526 <*package>

```

## 9.1 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.

```

3527 <<*More package options>> ≡
3528 \DeclareOption{safe=none}{\let\bbl@opt@safe\@empty}
3529 \DeclareOption{safe=bib}{\def\bbl@opt@safe{B}}
3530 \DeclareOption{safe=ref}{\def\bbl@opt@safe{R}}
3531 <</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.

```

3532 \bbl@trace{Cross referencing macros}
3533 \ifx\bbl@opt@safe\@empty\else
3534   \def\@newl@bel#1#2#3{%
3535     {\@safe@activetrue
3536       \bbl@ifunset{#1@#2}%
3537         \relax
3538         {\gdef\@multiplelabels{%
3539           \@latex@warning@no@line{There were multiply-defined labels}}}%
3540           \@latex@warning@no@line{Label `#2' multiply defined}}}%
3541   \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.

```

3542 \CheckCommand*\@testdef[3]{%
3543   \def\reserved@a{#3}%
3544   \expandafter\ifx\csname#1@#2\endcsname\reserved@a
3545   \else
3546     \@tempswatrue
3547   \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.

```

3548 \def\@testdef#1#2#3{% TODO. With @samestring?
3549   \@safe@activetrue
3550   \expandafter\let\expandafter\bbl@tempa\csname #1@#2\endcsname
3551   \def\bbl@tempb{#3}%
3552   \@safe@activesfalse
3553   \ifx\bbl@tempa\relax
3554   \else
3555     \edef\bbl@tempa{\expandafter\strip@prefix\meaning\bbl@tempa}%
3556   \fi

```

```

3557 \edef\bbl@tempb{\expandafter\strip@prefix\meaning\bbl@tempb}%
3558 \ifx\bbl@tempa\bbl@tempb
3559 \else
3560 \@tempswatrue
3561 \fi}
3562 \fi

```

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

```

3563 \bbl@xin@{R}\bbl@opt@safe
3564 \ifin@
3565 \bbl@redefineroast\ref#1{%
3566 \@safe@activetrue\org@ref{#1}\@safe@activesfalse}
3567 \bbl@redefineroast\pageref#1{%
3568 \@safe@activetrue\org@pageref{#1}\@safe@activesfalse}
3569 \else
3570 \let\org@ref\ref
3571 \let\org@pageref\pageref
3572 \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.

```

3573 \bbl@xin@{B}\bbl@opt@safe
3574 \ifin@
3575 \bbl@redefine\@citex[#1]#2{%
3576 \@safe@activetrue\edef\@tempa{#2}\@safe@activesfalse
3577 \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.

```

3578 \AtBeginDocument{%
3579 \@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.)

```

3580 \def\@citex[#1][#2]#3{%
3581 \@safe@activetrue\edef\@tempa{#3}\@safe@activesfalse
3582 \org@@citex[#1][#2]{\@tempa}}%
3583 }{}

```

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

```

3584 \AtBeginDocument{%
3585 \@ifpackageloaded{cite}{%
3586 \def\@citex[#1]#2{%
3587 \@safe@activetrue\org@@citex[#1][#2]\@safe@activesfalse}%
3588 }{}

```

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

```

3589 \bbl@redefine\nocite#1{%
3590 \@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.

```
3591 \bbl@redefine\bibcite{%
3592   \bbl@cite@choice
3593   \bibcite}
```

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

```
3594 \def\bbl@bibcite#1#2{%
3595   \org@bibcite{#1}{\@safe@activesfalse#2}}
```

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

```
3596 \def\bbl@cite@choice{%
3597   \global\let\bibcite\bbl@bibcite
3598   \@ifpackageloaded{natbib}{\global\let\bibcite\org@bibcite}{}%
3599   \@ifpackageloaded{cite}{\global\let\bibcite\org@bibcite}{}%
3600   \global\let\bbl@cite@choice\relax}
```

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

```
3601 \AtBeginDocument{\bbl@cite@choice}
```

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

```
3602 \bbl@redefine\@bibitem#1{%
3603   \@safe@activetrue\org@@bibitem{#1}\@safe@activesfalse}
3604 \else
3605   \let\org@nocite\nocite
3606   \let\org@@citex\@citex
3607   \let\org@bibcite\bibcite
3608   \let\org@@bibitem\@bibitem
3609 \fi
```

## 9.2 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.

```
3610 \bbl@trace{Marks}
3611 \IfBabelLayout{sectioning}
3612   {\ifx\bbl@opt@headfoot\@nnil
3613     \g@addto@macro\@resetactivechars{%
3614       \set@typeset@protect
3615       \expandafter\select@language@x\expandafter{\bbl@main@language}%
3616       \let\protect\noexpand
3617       \ifcase\bbl@bidimode\else % Only with bidi. See also above
3618         \edef\thepage{%
3619           \noexpand\babelsublr{\unexpanded\expandafter{\thepage}}}%
3620       \fi}%
3621   }
```



```

3621 \fi}
3622 {\ifbbl@single\else
3623 \bbl@ifunset{markright } \bbl@redefine\bbl@redefineroobust
3624 \markright#1{%
3625 \bbl@ifblank{#1}%
3626 {\org@markright{}}%
3627 {\toks@{#1}%
3628 \bbl@exp{%
3629 \\\org@markright{\\\protect\\foreignlanguage{\language}%
3630 {\\\protect\\bbl@restore@actives\the\toks@}}}%

```

`\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, L<sup>A</sup>T<sub>E</sub>X stores the definition in an intermediate macro, so it's not necessary anymore, but it's preserved for older versions.)

```

3631 \ifx\@mkboth\markboth
3632 \def\bbl@tempc{\let\@mkboth\markboth}
3633 \else
3634 \def\bbl@tempc{
3635 \fi
3636 \bbl@ifunset{markboth } \bbl@redefine\bbl@redefineroobust
3637 \markboth#1#2{%
3638 \protected@edef\bbl@tempb##1{%
3639 \protect\foreignlanguage
3640 {\language}{\protect\bbl@restore@actives##1}}%
3641 \bbl@ifblank{#1}%
3642 {\toks@{}}%
3643 {\toks@\expandafter{\bbl@tempb{#1}}}%
3644 \bbl@ifblank{#2}%
3645 {\@temptokena{}}%
3646 {\@temptokena\expandafter{\bbl@tempb{#2}}}%
3647 \bbl@exp{\\\org@markboth{\the\toks@}{\the\@temptokena}}
3648 \bbl@tempc
3649 \fi} % end ifbbl@single, end \IfBabelLayout

```

## 9.3 Preventing clashes with other packages

### 9.3.1 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.

```

3650 \bbl@trace{Preventing clashes with other packages}
3651 \bbl@xin@{R}\bbl@opt@safe

```

```

3652 \ifin@
3653 \AtBeginDocument{%
3654   \ifpackageloaded{ifthen}{%
3655     \bbl@redefine@long\ifthenelse#1#2#3{%
3656       \let\bbl@temp@pref\pageref
3657       \let\pageref\org@pageref
3658       \let\bbl@temp@ref\ref
3659       \let\ref\org@ref
3660       \@safe@activetrue
3661       \org@ifthenelse{#1}%
3662       {\let\pageref\bbl@temp@pref
3663        \let\ref\bbl@temp@ref
3664        \@safe@activetrue
3665        #2}%
3666       {\let\pageref\bbl@temp@pref
3667        \let\ref\bbl@temp@ref
3668        \@safe@activetrue
3669        #3}%
3670     }%
3671   }{}%
3672 }

```

### 9.3.2 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 `\vrefpagemum`.

```

3673 \AtBeginDocument{%
3674   \ifpackageloaded{varioref}{%
3675     \bbl@redefine\@@vpageref#1[#2]#3{%
3676       \@safe@activetrue
3677       \org@@@vpageref{#1}[#2]{#3}%
3678       \@safe@activetrue}%
3679     \bbl@redefine\vrefpagemum#1#2{%
3680       \@safe@activetrue
3681       \org@vrefpagemum{#1}{#2}%
3682       \@safe@activetrue}%

```

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.

```

3683   \expandafter\def\csname Ref \endcsname#1{%
3684     \protected@edef\@tempa{\org@ref{#1}}\expandafter\MakeUppercase\@tempa}
3685   }{}%
3686 }
3687 \fi

```

### 9.3.3 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.

```

3688 \AtEndOfPackage{%
3689   \AtBeginDocument{%

```

```

3690 \ifpackageloaded{hhline}%
3691   {\expandafter\ifx\csname normal@char\string\endcsname\relax
3692     \else
3693       \makeatletter
3694       \def\@currname{hhline}\input{hhline.sty}\makeatother
3695       \fi}%
3696   {}}

```

`\substitutefontfamily` Deprecated. Use the tools provides by  $\TeX$ . 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.

```

3697 \def\substitutefontfamily#1#2#3{%
3698   \lowercase{\immediate\openout15=#1#2.fd\relax}%
3699   \immediate\write15{%
3700     \string\ProvidesFile{#1#2.fd}%
3701     [\the\year/\two@digits{\the\month}/\two@digits{\the\day}
3702     \space generated font description file]^^J
3703     \string\DeclareFontFamily{#1}{#2}{ }^^J
3704     \string\DeclareFontShape{#1}{#2}{m}{n}{<->ssub * #3/m/n}{ }^^J
3705     \string\DeclareFontShape{#1}{#2}{m}{it}{<->ssub * #3/m/it}{ }^^J
3706     \string\DeclareFontShape{#1}{#2}{m}{sl}{<->ssub * #3/m/sl}{ }^^J
3707     \string\DeclareFontShape{#1}{#2}{m}{sc}{<->ssub * #3/m/sc}{ }^^J
3708     \string\DeclareFontShape{#1}{#2}{b}{n}{<->ssub * #3/bx/n}{ }^^J
3709     \string\DeclareFontShape{#1}{#2}{b}{it}{<->ssub * #3/bx/it}{ }^^J
3710     \string\DeclareFontShape{#1}{#2}{b}{sl}{<->ssub * #3/bx/sl}{ }^^J
3711     \string\DeclareFontShape{#1}{#2}{b}{sc}{<->ssub * #3/bx/sc}{ }^^J
3712   }%
3713   \closeout15
3714 }
3715 \@onlypreamble\substitutefontfamily

```

## 9.4 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. Requested encodings are currently stored in `\@fontenc@load@list`. 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`

```

3716 \bbl@trace{Encoding and fonts}
3717 \newcommand\BabelNonASCII{LGR,X2,OT2,OT3,OT6,LHE,LWN,LMA,LMC,LMS,LMU}
3718 \newcommand\BabelNonText{TS1,T3,TS3}
3719 \let\org@TeX\TeX
3720 \let\org@LaTeX\LaTeX
3721 \let\ensureascii\@firstofone
3722 \AtBeginDocument{%
3723   \def\@elt#1{, #1,}%
3724   \edef\bbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3725   \let\@elt\relax
3726   \let\bbl@tempb\@empty
3727   \def\bbl@tempc{OT1}%
3728   \bbl@foreach\BabelNonASCII{% LGR loaded in a non-standard way
3729     \bbl@ifunset{T@#1}{ }\def\bbl@tempb{#1}}%
3730   \bbl@foreach\bbl@tempa{%
3731     \bbl@xin@{#1}{\BabelNonASCII}%
3732     \ifin@
3733       \def\bbl@tempb{#1}% Store last non-ascii

```

```

3734 \else\bblexin@{#1}{\BabelNonText}% Pass
3735 \ifin@else
3736 \def\bbbl@tempc{#1}% Store last ascii
3737 \fi
3738 \fi}%
3739 \ifx\bbbl@tempb\@empty\else
3740 \bblexin@{,\cf@encoding,}{,\BabelNonASCII,\BabelNonText,}%
3741 \ifin@else
3742 \edef\bbbl@tempc{\cf@encoding}% The default if ascii wins
3743 \fi
3744 \edef\ensureascii#1{%
3745 {\noexpand\fontencoding{\bbbl@tempc}\noexpand\selectfont#1}}%
3746 \DeclareTextCommandDefault{\TeX}{\ensureascii{\org@TeX}}%
3747 \DeclareTextCommandDefault{\LaTeX}{\ensureascii{\org@LaTeX}}%
3748 \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.

```

3749 \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.

```

3750 \AtBeginDocument{%
3751 \ifpackageloaded{fontspec}%
3752 {\xdef\latinencoding{%
3753 \ifx\UTFencname\@undefined
3754 EU\ifcase\bbbl@engine\or2\or1\fi
3755 \else
3756 \UTFencname
3757 \fi}}%
3758 {\gdef\latinencoding{OT1}%
3759 \ifx\cf@encoding\bbbl@t@one
3760 \xdef\latinencoding{\bbbl@t@one}%
3761 \else
3762 \def\@elt#1{,#1,}%
3763 \edef\bbbl@tempa{\expandafter\@gobbletwo\@fontenc@load@list}%
3764 \let\@elt\relax
3765 \bblexin@{,T1,}\bbbl@tempa
3766 \ifin@
3767 \xdef\latinencoding{\bbbl@t@one}%
3768 \fi
3769 \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.

```

3770 \DeclareRobustCommand{\latintext}{%
3771 \fontencoding{\latinencoding}\selectfont
3772 \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.

```

3773 \ifx\@undefined\DeclareTextFontCommand
3774 \DeclareRobustCommand{\textlatin}[1]{\leavevmode{\latintext #1}}

```

```

3775 \else
3776   \DeclareTextFontCommand{\textlatin}{\latintext}
3777 \fi

```

For several functions, we need to execute some code with `\selectfont`. With  $\text{\LaTeX}$  2021-06-01, there is a hook for this purpose, but in older versions the  $\text{\LaTeX}$  command is patched (the latter solution will be eventually removed).

```

3778 \bbl@ifformatlater{2021-06-01}%
3779   {\def\bbl@patchfont#1{\AddToHook{selectfont}{#1}}}
3780   {\def\bbl@patchfont#1{%
3781     \expandafter\bbl@add\csname selectfont \endcsname{#1}%
3782     \expandafter\bbl@to\global\csname selectfont \endcsname}}

```

## 9.5 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 `ARAB` (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  $\text{\TeX}$  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 $\text{\TeX}$ -ja` shows, vertical typesetting is possible, too.

```

3783 \bbl@trace{Loading basic (internal) bidi support}
3784 \ifodd\bbl@engine
3785 \else % TODO. Move to txtbabel
3786   \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
3787     \bbl@error
3788     {The bidi method 'basic' is available only in\%
3789     luatex. I'll continue with 'bidi=default', so\%
3790     expect wrong results}%
3791     {See the manual for further details.}%
3792     \let\bbl@beforeforeign\leavevmode
3793     \AtEndOfPackage{%
3794       \EnableBabelHook{babel-bidi}%
3795       \bbl@xebidipar}
3796   \fi\fi
3797   \def\bbl@loadxebidi#1{%
3798     \ifx\RTLfootnotetext\@undefined
3799       \AtEndOfPackage{%
3800         \EnableBabelHook{babel-bidi}%
3801         \ifx\fontspec\@undefined
3802           \bbl@loadfontspec % bidi needs fontspec
3803         \fi
3804         \usepackage#1{bidi}}%
3805     \fi}

```

```

3806 \ifnum\bbbl@bidimode>200
3807 \ifcase\expandafter\@gobbletwo\the\bbbl@bidimode\or
3808 \bbbl@tentative{bidi=bidi}
3809 \bbbl@loadxebidi{}
3810 \or
3811 \bbbl@loadxebidi{[rldocument]}
3812 \or
3813 \bbbl@loadxebidi{}
3814 \fi
3815 \fi
3816 \fi
3817 % TODO? Separate:
3818 \ifnum\bbbl@bidimode=\@ne
3819 \let\bbbl@beforeforeign\leavevmode
3820 \ifodd\bbbl@engine
3821 \newattribute\bbbl@attr@dir
3822 \directlua{ Babel.attr_dir = luatexbase.registernumber'bbbl@attr@dir' }
3823 \bbbl@exp{\output{\bodydir\pagedir\the\output}}
3824 \fi
3825 \AtEndOfPackage{%
3826 \EnableBabelHook{babel-bidi}%
3827 \ifodd\bbbl@engine\else
3828 \bbbl@xebidipar
3829 \fi}
3830 \fi

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

3831 \bbbl@trace{Macros to switch the text direction}
3832 \def\bbbl@alscripts{,Arabic,Syriac,Thaana,}
3833 \def\bbbl@rscripts{% TODO. Base on codes ??
3834 ,Imperial Aramaic,Avestan,Cypriot,Hatran,Hebrew,%
3835 Old Hungarian,Old Hungarian,Lydian,Mandaean,Manichaeen,%
3836 Manichaeen,Meroitic Cursive,Meroitic,Old North Arabian,%
3837 Nabataean,N'Ko,Orkhon,Palmyrene,Inscriptional Pahlavi,%
3838 Psalter Pahlavi,Phoenician,Inscriptional Parthian,Samaritan,%
3839 Old South Arabian,}%
3840 \def\bbbl@provide@dirs#1{%
3841 \bbbl@xin@{\csname bbl@sname@#1\endcsname}{\bbbl@alscripts\bbbl@rscripts}%
3842 \ifin@
3843 \global\bbbl@csarg\chardef{wdir@#1}\@ne
3844 \bbbl@xin@{\csname bbl@sname@#1\endcsname}{\bbbl@alscripts}%
3845 \ifin@
3846 \global\bbbl@csarg\chardef{wdir@#1}\tw@ % useless in xetex
3847 \fi
3848 \else
3849 \global\bbbl@csarg\chardef{wdir@#1}\z@
3850 \fi
3851 \ifodd\bbbl@engine
3852 \bbbl@csarg\ifcase{wdir@#1}%
3853 \directlua{ Babel.locale_props[\the\localeid].textdir = 'l' }%
3854 \or
3855 \directlua{ Babel.locale_props[\the\localeid].textdir = 'r' }%
3856 \or
3857 \directlua{ Babel.locale_props[\the\localeid].textdir = 'al' }%
3858 \fi
3859 \fi}
3860 \def\bbbl@switchdir{%
3861 \bbbl@ifunset{bbl@lsys@\languagename}{\bbbl@provide@lsys{\languagename}}{}}%

```

```

3862 \bbl@ifunset{bbl@wdir@\languagename}{\bbl@provide@dirs{\languagename}}{}}%
3863 \bbl@exp{\bbl@setdirs\bbl@ccl{wdir}}}%
3864 \def\bbl@setdirs#1{% TODO - math
3865 \ifcase\bbl@select@type % TODO - strictly, not the right test
3866 \bbl@bodydir{#1}%
3867 \bbl@pardir{#1}%
3868 \fi
3869 \bbl@textdir{#1}}
3870 % TODO. Only if \bbl@bidimode > 0?:
3871 \AddBabelHook{babel-bidi}{afterextras}{\bbl@switchdir}
3872 \DisableBabelHook{babel-bidi}

```

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

```

3873 \ifodd\bbl@engine % luatex=1
3874 \else % pdftex=0, xetex=2
3875 \newcount\bbl@dirlevel
3876 \chardef\bbl@thetextdir\z@
3877 \chardef\bbl@thepardir\z@
3878 \def\bbl@textdir#1{%
3879 \ifcase#1\relax
3880 \chardef\bbl@thetextdir\z@
3881 \bbl@textdir@i\beginL\endL
3882 \else
3883 \chardef\bbl@thetextdir@ne
3884 \bbl@textdir@i\beginR\endR
3885 \fi}
3886 \def\bbl@textdir@i#1#2{%
3887 \ifhmode
3888 \ifnum\currentgrouplevel>\z@
3889 \ifnum\currentgrouplevel=\bbl@dirlevel
3890 \bbl@error{Multiple bidi settings inside a group}%
3891 {I'll insert a new group, but expect wrong results.}%
3892 \bgroup\aftergroup#2\aftergroup\egroup
3893 \else
3894 \ifcase\currentgrouptype\or % 0 bottom
3895 \aftergroup#2% 1 simple {}
3896 \or
3897 \bgroup\aftergroup#2\aftergroup\egroup % 2 hbox
3898 \or
3899 \bgroup\aftergroup#2\aftergroup\egroup % 3 adj hbox
3900 \or\or\or % vbox vtop align
3901 \or
3902 \bgroup\aftergroup#2\aftergroup\egroup % 7 noalign
3903 \or\or\or\or\or\or % output math disc insert vcent mathchoice
3904 \or
3905 \aftergroup#2% 14 \begingroup
3906 \else
3907 \bgroup\aftergroup#2\aftergroup\egroup % 15 adj
3908 \fi
3909 \fi
3910 \bbl@dirlevel\currentgrouplevel
3911 \fi
3912 #1%
3913 \fi}
3914 \def\bbl@pardir#1{\chardef\bbl@thepardir#1\relax}
3915 \let\bbl@bodydir@gobble
3916 \let\bbl@pagedir@gobble
3917 \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).

```

3918 \def\bbl@xebidipar{%
3919   \let\bbl@xebidipar\relax
3920   \TeXeTstate\@ne
3921   \def\bbl@xeeverypar{%
3922     \ifcase\bbl@thepardir
3923       \ifcase\bbl@thetextdir\else\beginR\fi
3924     \else
3925       {\setbox\z@\lastbox\beginR\box\z@}%
3926     \fi}%
3927   \let\bbl@severypar\everypar
3928   \newtoks\everypar
3929   \everypar=\bbl@severypar
3930   \bbl@severypar{\bbl@xeeverypar\the\everypar}}
3931 \ifnum\bbl@bidimode>200
3932   \let\bbl@textdir@i\@gobbletwo
3933   \let\bbl@xebidipar\@empty
3934   \AddBabelHook{bidi}{foreign}{%
3935     \def\bbl@tempa{\def\BabelText####1}%
3936     \ifcase\bbl@thetextdir
3937       \expandafter\bbl@tempa\expandafter{\BabelText{\LR{##1}}}%
3938     \else
3939       \expandafter\bbl@tempa\expandafter{\BabelText{\RL{##1}}}%
3940     \fi}
3941   \def\bbl@pardir#1{\ifcase#1\relax\setLR\else\setRL\fi}
3942 \fi
3943 \fi

```

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

```

3944 \DeclareRobustCommand\babelsublr[1]{\leavevmode\bbl@textdir\z@#1}
3945 \AtBeginDocument{%
3946   \ifx\pdfstringdefDisableCommands\@undefined\else
3947     \ifx\pdfstringdefDisableCommands\relax\else
3948       \pdfstringdefDisableCommands{\let\babelsublr\@firstofone}%
3949     \fi
3950   \fi}

```

## 9.6 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.

```

3951 \bbl@trace{Local Language Configuration}
3952 \ifx\loadlocalcfg\@undefined
3953   \ifpackagewith{babel}{noconfigs}%
3954     {\let\loadlocalcfg\@gobble}%
3955     {\def\loadlocalcfg#1{%
3956       \InputIfFileExists{#1.cfg}%
3957       {\typeout{*****^J%
3958                 * Local config file #1.cfg used^^J%
3959                 *}}}%
3960     \@empty}}
3961 \fi

```



## 9.7 Language options

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

```
3962 \bbl@trace{Language options}
3963 \let\bbl@afterlang\relax
3964 \let\BabelModifiers\relax
3965 \let\bbl@loaded@empty
3966 \def\bbl@load@language#1{%
3967   \InputIfFileExists{#1.ldf}%
3968   {\edef\bbl@loaded{\CurrentOption
3969     \ifx\bbl@loaded@empty\else,\bbl@loaded\fi}%
3970     \expandafter\let\expandafter\bbl@afterlang
3971     \csname\CurrentOption.ldf-h@@k\endcsname
3972     \expandafter\let\expandafter\BabelModifiers
3973     \csname bbl@mod@\CurrentOption\endcsname}%
3974   {\bbl@error{%
3975     Unknown option '\CurrentOption'. Either you misspelled it\\
3976     or the language definition file \CurrentOption.ldf was not found}}%
3977   Valid options are, among others: shorthands=, KeepShorthandsActive,\\
3978   activeacute, activegrave, noconfigs, safe=, main=, math=\\
3979   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.

```
3980 \def\bbl@try@load@lang#1#2#3{%
3981   \IfFileExists{\CurrentOption.ldf}%
3982   {\bbl@load@language{\CurrentOption}}%
3983   {#1\bbl@load@language{#2}#3}}
3984 %
3985 \DeclareOption{hebrew}{%
3986   \input{rlbabel.def}%
3987   \bbl@load@language{hebrew}}
3988 \DeclareOption{hungarian}{\bbl@try@load@lang{}{magyar}{}}
3989 \DeclareOption{lowersorbian}{\bbl@try@load@lang{}{lsorbian}{}}
3990 \DeclareOption{nynorsk}{\bbl@try@load@lang{}{norsk}{}}
3991 \DeclareOption{polutonikogreek}{%
3992   \bbl@try@load@lang{}{greek}{\languageattribute{greek}{polutoniko}}}
3993 \DeclareOption{russian}{\bbl@try@load@lang{}{russianb}{}}
3994 \DeclareOption{ukrainian}{\bbl@try@load@lang{}{ukraineb}{}}
3995 \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.

```
3996 \ifx\bbl@opt@config@nnil
3997   \@ifpackagewith{babel}{noconfigs}{}%
3998   {\InputIfFileExists{bblopts.cfg}%
3999     {\typeout{*****^J%
4000       * Local config file bblopts.cfg used^^J%
4001       *}}%
4002     {}}%
4003 \else
4004   \InputIfFileExists{\bbl@opt@config.cfg}%
4005   {\typeout{*****^J%
4006     * Local config file \bbl@opt@config.cfg used^^J%
```

```

4007         *}}%
4008     {\bbl@error{%
4009         Local config file '\bbl@opt@config.cfg' not found}{%
4010         Perhaps you misspelled it.}}%
4011 \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 existing languages (note this list also contains the language given with `main` as the last element). If not declared above, the names of the option and the file are the same. There are two steps – first process option names and collect the result, which then do the actual declarations.

To allow multiple overlapping replacements, commas in `\bbl@language@opts` are doubled.

```

4012 \let\bbl@elt\relax
4013 \let\bbl@tempe\@empty
4014 \bbl@foreach\@classoptionslist{%
4015     \bbl@xin@{,#1,$}{\bbl@language@opts$}% Match last
4016     \ifin@%else
4017         \bbl@xin@{,#1,}{\bbl@language@opts$}% Match non-last
4018         \ifin@
4019             \bbl@replace\bbl@language@opts{,#1,}{,,}%
4020             \edef\bbl@tempe{\bbl@tempe\bbl@elt{3}{#1}}%
4021         %else
4022             \babel@savecnt\z@ % Use as temp
4023             \ifnum\bbl@iniflag<\thr@@ % Optimization: 3 = always ini
4024                 \IfFileExists{#1.ldf}{\advance\babel@savecnt\@ne}{}%
4025             %fi
4026             \ifnum\bbl@iniflag>\z@ % Optimization: 0 = always ldf
4027                 \IfFileExists{babel-#1.tex}{\advance\babel@savecnt\tw@}{}%
4028             %fi
4029             \ifnum\babel@savecnt>\z@
4030                 \edef\bbl@tempe{\bbl@tempe\bbl@elt{\the\babel@savecnt}{#1}}%
4031             %fi
4032         %fi
4033     \fi}
4034 %
4035 \let\bbl@savemain\@empty
4036 \bbl@foreach\bbl@language@opts{%
4037     \edef\bbl@tempe{\bbl@tempe\bbl@elt{3}{#1}}%
4038     \def\bbl@elt#1#2#3{%
4039         \ifx#3\relax % if last
4040             \bbl@ifunset{ds@#2}{}%
4041             {\bbl@exp{\def\\bbl@savemain{\\DeclareOption{#2}{\[ds@#2]}}}%
4042             \bbl@add\bbl@savemain{\bbl@elt{#1}{#2}}% Save main
4043             \DeclareOption{#2}{}%
4044         %else
4045             \ifnum\bbl@iniflag<\tw@ % other as ldf
4046                 \ifodd#1\relax % Class: if ldf exists 1,3. Package: always 3
4047                     \bbl@ifunset{ds@#2}{%
4048                         {\DeclareOption{#2}{\bbl@load@language{#2}}}%
4049                     }%
4050                 %fi
4051             %else % other as ini
4052                 \ifnum#1>\@ne % % Class: if ini exists 2,3. Package: always 3
4053                     \DeclareOption{#2}{%
4054                         \bbl@ldfinit
4055                         \babelprovide[import]{#2}%
4056                         \bbl@afterldf{}}%
4057                 %fi

```

```

4058 \fi
4059 \fi
4060 #3}
4061 \bbl@tempe\relax % \relax catches last

```

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.

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

```

4062 \def\AfterBabelLanguage#1{%
4063 \bbl@ifsamestring\CurrentOption{#1}{\global\bbl@add\bbl@afterlang}{}}
4064 \DeclareOption*{}
4065 \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.

```

4066 \bbl@trace{Option 'main'}
4067 \ifx\bbl@opt@main\@nnil
4068 \edef\bbl@tempa{\@classoptionslist,\bbl@language@opts}
4069 \let\bbl@tempc\@empty
4070 \bbl@for\bbl@tempb\bbl@tempa{%
4071 \bbl@xin@{\bbl@tempb,}{,\bbl@loaded,}%
4072 \ifin@{\edef\bbl@tempc{\bbl@tempb}\fi}
4073 \def\bbl@tempa#1,#2\@nnil{\def\bbl@tempb{#1}}
4074 \expandafter\bbl@tempa\bbl@loaded,\@nnil
4075 \ifx\bbl@tempb\bbl@tempc\else
4076 \bbl@warning{%
4077 Last declared language option is '\bbl@tempc',\%
4078 but the last processed one was '\bbl@tempb'.\%
4079 The main language can't be set as both a global\%
4080 and a package option. Use 'main=\bbl@tempc' as\%
4081 option. Reported}%
4082 \fi
4083 \fi
4084 \def\bbl@elt#1#2{% main
4085 \ifodd\bbl@iniflag % as ini = 1(=), 3(*=)
4086 \ifnum#1>\@ne % % Class: if ini exists 2,3. Package: always 3
4087 \def\CurrentOption{#2}% Directly, because luatexbase
4088 \bbl@ldfinit
4089 \babelprovide[\bbl@opt@provide,main,import]{#2}%
4090 \bbl@afterldf}%
4091 \fi
4092 \else % as ldf = 0(no), 2(+=)
4093 \ifodd#1\relax % Class: if ldf exists 1,3. Package: always 3
4094 \bbl@ifunset{ds@#2}%
4095 {\DeclareOption{#2}{\bbl@load@language{#2}}}%
4096 {}%
4097 \ExecuteOptions{#2}%
4098 \DeclareOption*{}%
4099 \ProcessOptions%
4100 \fi
4101 \fi}
4102 \bbl@savemain
4103 \def\AfterBabelLanguage%

```

```

4104 \bbl@error
4105 {Too late for \string\AfterBabelLanguage}%
4106 {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.

```

4107 \ifx\bbl@main@language\@undefined
4108 \bbl@info{%
4109   You haven't specified a language. I'll use 'nil'\%
4110   as the main language. Reported}
4111 \bbl@load@language{nil}
4112 \fi
4113 \</package>

```

## 10 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.

A proxy file for `switch.def`

```

4114 \<*kernel>
4115 \let\bbl@onlyswitch\@empty
4116 \input babel.def
4117 \let\bbl@onlyswitch\@undefined
4118 \</kernel>
4119 \<*patterns>

```

## 11 Loading hyphenation patterns

The following code is meant to be read by `ini $\TeX$`  because it should instruct  $\TeX$  to read hyphenation patterns. To this end the `docstrip` option `patterns` is used to include this code in the file `hyphen.cfg`. Code is written with lower level macros.

```

4120 \<<Make sure ProvidesFile is defined>>
4121 \ProvidesFile{hyphen.cfg}[\<<date>>] [\<<version>>] Babel hyphens]
4122 \xdef\bbl@format{\jobname}
4123 \def\bbl@version{\<<version>>}
4124 \def\bbl@date{\<<date>>}
4125 \ifx\AtBeginDocument\@undefined
4126 \def\@empty{}
4127 \fi
4128 \<<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.

```

4129 \def\process@line#1#2 #3 #4 {%
4130 \ifx=#1%
4131 \process@synonym{#2}%

```

```

4132 \else
4133 \process@language{#1#2}{#3}{#4}%
4134 \fi
4135 \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.

```

4136 \toks@{}
4137 \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.

```

4138 \def\process@synonym#1{%
4139 \ifnum\last@language=\m@ne
4140 \toks@\expandafter{\the\toks@\relax\process@synonym{#1}}%
4141 \else
4142 \expandafter\chardef\csname l@#1\endcsname\last@language
4143 \wlog{\string\l@#1=\string\language\the\last@language}%
4144 \expandafter\let\csname #1hyphenmins\expandafter\endcsname
4145 \csname\language\hyphenmins\endcsname
4146 \let\bbl@elt\relax
4147 \edef\bbl@languages{\bbl@languages\bbl@elt{#1}{\the\last@language}{}}%
4148 \fi}

```

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

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

For some hyphenation patterns it is needed to load them with a specific font encoding selected. This can be specified in the file `language.dat` by adding for instance ‘:T1’ to the name of the language.

The macro `\bbl@get@enc` extracts the font encoding from the language name and stores it in `\bbl@hyph@enc`. The latter can be used in hyphenation files if you need to set a behavior depending on the given encoding (it is set to empty if no encoding is given).

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

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

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

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

`\bbl@languages` saves a snapshot of the loaded languages in the form

`\bbl@elt{<language-name>}{<number>}{<patterns-file>}{<exceptions-file>}`. Note the last 2 arguments are empty in ‘dialects’ defined in `language.dat` with `=`. Note also the language name can have encoding info.

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

```

4149 \def\process@language#1#2#3{%
4150 \expandafter\addlanguage\csname l@#1\endcsname
4151 \expandafter\language\csname l@#1\endcsname
4152 \edef\language{#1}%
4153 \bbl@hook@everylanguage{#1}%

```

```

4154 % > luatex
4155 \bbl@get@enc#1::\@@@
4156 \begingroup
4157   \lefthyphenmin\m@ne
4158   \bbl@hook@loadpatterns{#2}%
4159   % > luatex
4160   \ifnum\lefthyphenmin=\m@ne
4161   \else
4162     \expandafter\xdef\csname #1hyphenmins\endcsname{%
4163       \the\lefthyphenmin\the\righthyphenmin}%
4164   \fi
4165 \endgroup
4166 \def\bbl@tempa{#3}%
4167 \ifx\bbl@tempa\@empty\else
4168   \bbl@hook@loadexceptions{#3}%
4169   % > luatex
4170   \fi
4171   \let\bbl@elt\relax
4172   \edef\bbl@languages{%
4173     \bbl@languages\bbl@elt{#1}{\the\language}{#2}{\bbl@tempa}}%
4174   \ifnum\the\language=\z@
4175     \expandafter\ifx\csname #1hyphenmins\endcsname\relax
4176       \set@hyphenmins\tw@\thr@@\relax
4177     \else
4178       \expandafter\expandafter\expandafter\set@hyphenmins
4179       \csname #1hyphenmins\endcsname
4180     \fi
4181     \the\toks@
4182     \toks@{}%
4183   \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.

```

4184 \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.

```

4185 \def\bbl@hook@everylanguage#1{}
4186 \def\bbl@hook@loadpatterns#1{\input #1\relax}
4187 \let\bbl@hook@loadexceptions\bbl@hook@loadpatterns
4188 \def\bbl@hook@loadkernel#1{%
4189   \def\addlanguage{\csname newlanguage\endcsname}%
4190   \def\adddialect##1##2{%
4191     \global\chardef##1##2\relax
4192     \wlog{\string##1 = a dialect from \string\language##2}}%
4193   \def\iflanguage##1{%
4194     \expandafter\ifx\csname l@##1\endcsname\relax
4195       \@nolanerr{##1}%
4196     \else
4197       \ifnum\csname l@##1\endcsname=\language
4198         \expandafter\expandafter\expandafter\@firstoftwo
4199       \else
4200         \expandafter\expandafter\expandafter\@secondoftwo
4201       \fi
4202     \fi}%
4203   \def\providehyphenmins##1##2{%
4204     \expandafter\ifx\csname ##1hyphenmins\endcsname\relax

```

```

4205 \namedef{##1hyphenmins}{##2}%
4206 \fi}%
4207 \def\set@hyphenmins##1##2{%
4208 \lefthyphenmin##1\relax
4209 \righthyphenmin##2\relax}%
4210 \def\selectlanguage{%
4211 \errhelp{Selecting a language requires a package supporting it}%
4212 \errmessage{Not loaded}}%
4213 \let\foreignlanguage\selectlanguage
4214 \let\otherlanguage\selectlanguage
4215 \expandafter\let\csname otherlanguage*\endcsname\selectlanguage
4216 \def\bbl@usehooks##1##2{% TODO. Temporary!!
4217 \def\setlocale{%
4218 \errhelp{Find an armchair, sit down and wait}%
4219 \errmessage{Not yet available}}%
4220 \let\uselocale\setlocale
4221 \let\locale\setlocale
4222 \let\selectlocale\setlocale
4223 \let\localename\setlocale
4224 \let\textlocale\setlocale
4225 \let\textlanguage\setlocale
4226 \let\languagegettext\setlocale}
4227 \begingroup
4228 \def\AddBabelHook#1#2{%
4229 \expandafter\ifx\csname bbl@hook@#2\endcsname\relax
4230 \def\next{\toks1}%
4231 \else
4232 \def\next{\expandafter\gdef\csname bbl@hook@#2\endcsname####1}%
4233 \fi
4234 \next}
4235 \ifx\directlua\undefined
4236 \ifx\XeTeXinputencoding\undefined\else
4237 \input xebabel.def
4238 \fi
4239 \else
4240 \input luababel.def
4241 \fi
4242 \openin1 = babel-\bbl@format.cfg
4243 \ifeof1
4244 \else
4245 \input babel-\bbl@format.cfg\relax
4246 \fi
4247 \closein1
4248 \endgroup
4249 \bbl@hook@loadkernel{switch.def}

```

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

```

4250 \openin1 = language.dat

```

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

```

4251 \def\languagename{english}%
4252 \ifeof1
4253 \message{I couldn't find the file language.dat,\space
4254 I will try the file hyphen.tex}
4255 \input hyphen.tex\relax
4256 \chardef\l@english\z@
4257 \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`.

```
4258 \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.

```
4259 \loop
4260 \endlinechar\m@ne
4261 \read1 to \bbl@line
4262 \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.

```
4263 \if T\ifeof1F\fi T\relax
4264 \ifx\bbl@line\@empty\else
4265 \edef\bbl@line{\bbl@line\space\space\space}%
4266 \expandafter\process@line\bbl@line\relax
4267 \fi
4268 \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.

```
4269 \begingroup
4270 \def\bbl@elt#1#2#3#4{%
4271 \global\language=#2\relax
4272 \gdef\language#1}%
4273 \def\bbl@elt##1##2##3##4{}}%
4274 \bbl@languages
4275 \endgroup
4276 \fi
4277 \closein1
```

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

```
4278 \if/\the\toks@\else
4279 \errhelp{language.dat loads no language, only synonyms}
4280 \errmessage{Orphan language synonym}
4281 \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.

```
4282 \let\bbl@line\@undefined
4283 \let\process@line\@undefined
4284 \let\process@synonym\@undefined
4285 \let\process@language\@undefined
4286 \let\bbl@get@enc\@undefined
4287 \let\bbl@hyph@enc\@undefined
4288 \let\bbl@tempa\@undefined
4289 \let\bbl@hook@loadkernel\@undefined
4290 \let\bbl@hook@everylanguage\@undefined
4291 \let\bbl@hook@loadpatterns\@undefined
4292 \let\bbl@hook@loadexceptions\@undefined
4293 </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].

```
4294 <<*More package options>> ≡
4295 \chardef\bbl@bidimode\z@
4296 \DeclareOption{bidi=default}{\chardef\bbl@bidimode=\@ne}
4297 \DeclareOption{bidi=basic}{\chardef\bbl@bidimode=101 }
4298 \DeclareOption{bidi=basic-r}{\chardef\bbl@bidimode=102 }
4299 \DeclareOption{bidi=bidi}{\chardef\bbl@bidimode=201 }
4300 \DeclareOption{bidi=bidi-r}{\chardef\bbl@bidimode=202 }
4301 \DeclareOption{bidi=bidi-l}{\chardef\bbl@bidimode=203 }
4302 <</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.

```
4303 <<*Font selection>> ≡
4304 \bbl@trace{Font handling with fontspec}
4305 \ifx\ExplSyntaxOn\@undefined\else
4306   \ExplSyntaxOn
4307   \catcode\ =10
4308   \def\bbl@loadfontspec{%
4309     \usepackage{fontspec}% TODO. Apply patch always
4310     \expandafter
4311     \def\csname msg-text->~fontspec/language-not-exist\endcsname##1##2##3##4{%
4312       Font '\l_fontspec_fontname_tl' is using the\\%
4313       default features for language '##1'.\\%
4314       That's usually fine, because many languages\\%
4315       require no specific features, but if the output is\\%
4316       not as expected, consider selecting another font.}
4317     \expandafter
4318     \def\csname msg-text->~fontspec/no-script\endcsname##1##2##3##4{%
4319       Font '\l_fontspec_fontname_tl' is using the\\%
4320       default features for script '##2'.\\%
4321       That's not always wrong, but if the output is\\%
4322       not as expected, consider selecting another font.}}
4323   \ExplSyntaxOff
4324 \fi
4325 \@onlypreamble\babelfont
4326 \newcommand\babelfont[2][]{% 1=langs/scripts 2=fam
4327   \bbl@foreach{#1}{%
4328     \expandafter\ifx\csname date##1\endcsname\relax
4329       \IfFileExists{babel-##1.tex}%
4330       {\babelprovide{##1}}}%
4331   }%
4332 \fi}%
4333 \edef\bbl@tempa{#1}%
4334 \def\bbl@tempb{#2}% Used by \bbl@bblfont
4335 \ifx\fontspec\@undefined
4336   \bbl@loadfontspec
4337 \fi
4338 \EnableBabelHook{babel-fontspec}% Just calls \bbl@switchfont
4339 \bbl@bblfont}
4340 \newcommand\bbl@bblfont[2][]{% 1=features 2=fontname, @font=rm|sf|tt
4341   \bbl@ifunset{\bbl@tempb family}%
```

```

4342 {\bbl@providfam{\bbl@tempb}}%
4343 {}%
4344 % For the default font, just in case:
4345 \bbl@ifunset{\bbl@lsys@\language}\bbl@provide@lsys{\language}}{}%
4346 \expandafter\bbl@ifblank\expandafter{\bbl@tempa}%
4347 {\bbl@csarg\edef{\bbl@tempb dflt@}{<>{#1}{#2}}}% save bbl@rmdflt@
4348 \bbl@exp{%
4349 \let\bbl@\bbl@tempb dflt@\language>\<\bbl@\bbl@tempb dflt@>%
4350 \bbl@font@set{\bbl@\bbl@tempb dflt@\language>%
4351 \<\bbl@tempb default>\<\bbl@tempb family>}}%
4352 {\bbl@foreach\bbl@tempa{% ie bbl@rmdflt@lang / *scrt
4353 \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:

```

4354 \def\bbl@providfam#1{%
4355 \bbl@exp{%
4356 \\\newcommand\<#1default>{}% Just define it
4357 \\\bbl@add@list\\bbl@font@fams{#1}%
4358 \\\DeclareRobustCommand\<#1family>{%
4359 \\\not@math@alphabet\<#1family>\relax
4360 % \\\prepare@family@series@update{#1}\<#1default>% TODO. Fails
4361 \\\fontfamily\<#1default>%
4362 \<ifx>\\\UseHooks\\\@undefined\<else>\\\UseHook{#1family}\<fi>%
4363 \\\selectfont}%
4364 \\\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.

```

4365 \def\bbl@nostdfont#1{%
4366 \bbl@ifunset{\bbl@WFF@\f@family}%
4367 {\bbl@csarg\gdef{\bbl@WFF@\f@family}}{}% Flag, to avoid dupl warns
4368 \bbl@infowarn{The current font is not a babel standard family:\\%
4369 #1%
4370 \fontname\font\\%
4371 There is nothing intrinsically wrong with this warning, and\\%
4372 you can ignore it altogether if you do not need these\\%
4373 families. But if they are used in the document, you should be\\%
4374 aware 'babel' will no set Script and Language for them, so\\%
4375 you may consider defining a new family with \string\babelfont.\\%
4376 See the manual for further details about \string\babelfont.\\%
4377 Reported}}
4378 {}}%
4379 \gdef\bbl@switchfont{%
4380 \bbl@ifunset{\bbl@lsys@\language}\bbl@provide@lsys{\language}}{}%
4381 \bbl@exp{% eg Arabic -> arabic
4382 \lowercase{\edef\\bbl@tempa{\bbl@cl{sname}}}}%
4383 \bbl@foreach\bbl@font@fams{%
4384 \bbl@ifunset{\bbl@##1dflt@\language}% (1) language?
4385 {\bbl@ifunset{\bbl@##1dflt@*\bbl@tempa}% (2) from script?
4386 {\bbl@ifunset{\bbl@##1dflt@}% 2=F - (3) from generic?
4387 {}% 123=F - nothing!
4388 {\bbl@exp{% 3=T - from generic
4389 \global\let\bbl@##1dflt@\language>%
4390 \<\bbl@##1dflt@>}}}%
4391 {\bbl@exp{% 2=T - from script
4392 \global\let\bbl@##1dflt@\language>%
4393 \<\bbl@##1dflt@*\bbl@tempa>}}}%
4394 {}}% 1=T - language, already defined
4395 \def\bbl@tempa{\bbl@nostdfont}}%

```

```

4396 \bbl@foreach\bbl@font@fams{%      don't gather with prev for
4397   \bbl@ifunset{\bbl@##1dflt@\language}%
4398   {\bbl@cs{\famrst@##1}%
4399   \global\bbl@csarg\let{\famrst@##1}\relax}%
4400   {\bbl@exp{% order is relevant. TODO: but sometimes wrong!
4401     \\bbl@add\\originalTeX{%
4402       \\bbl@font@rst{\bbl@cl{##1dflt}}}%
4403       \<##1default>\<##1family>{##1}}}%
4404     \\bbl@font@set{\<bbl@##1dflt@\language>% the main part!
4405       \<##1default>\<##1family>}}}%
4406 \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.

```

4407 \ifx\family\undefined\else      % if latex
4408 \ifcase\bbl@engine                % if pdftex
4409   \let\bbl@cckstdfonts\relax
4410 \else
4411   \def\bbl@cckstdfonts{%
4412     \begingroup
4413     \global\let\bbl@cckstdfonts\relax
4414     \let\bbl@tempa\@empty
4415     \bbl@foreach\bbl@font@fams{%
4416       \bbl@ifunset{\bbl@##1dflt@}%
4417       {\@nameuse{##1family}%
4418        \bbl@csarg\gdef{WFF@\family}}}% Flag
4419       \bbl@exp{\\bbl@add\\bbl@tempa{* \<##1family>= \family\\}%
4420         \space\space\fontname\font\\}%
4421       \bbl@csarg\xdef{##1dflt@}{\family}%
4422       \expandafter\xdef\csname ##1default\endcsname{\family}}}%
4423     }%
4424   \ifx\bbl@tempa\@empty\else
4425     \bbl@infowarn{The following font families will use the default\\%
4426       settings for all or some languages:\\%
4427       \bbl@tempa
4428       There is nothing intrinsically wrong with it, but\\%
4429       'babel' will no set Script and Language, which could\\%
4430       be relevant in some languages. If your document uses\\%
4431       these families, consider redefining them with \string\babelfont.\\%
4432       Reported}%
4433   \fi
4434 \endgroup}
4435 \fi
4436 \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.

```

4437 \def\bbl@font@set#1#2#3{% eg \bbl@rmdflt@lang \rmdefault \rmfamily
4438   \bbl@xin@{<>}{#1}%
4439   \ifin@
4440     \bbl@exp{\\bbl@fontspec@set\\#1\expandafter@gobbletwo#1\\#3}%
4441   \fi
4442   \bbl@exp{%
4443     \def\\#2{#1}%      eg, \rmdefault{\bbl@rmdflt@lang}
4444     \\bbl@ifsamestring{#2}{\family}%
4445     {\#3
4446      \\bbl@ifsamestring{\family}{\bfdefault}{\\bfseries}}}%

```

```

4447 \let\bbbl@tempa\relax}%
4448 {}}}
4449 % TODO - next should be global?, but even local does its job. I'm
4450 % still not sure -- must investigate:
4451 \def\bbbl@fontspec@set#1#2#3#4{% eg \bbbl@rmdflt@lang fnt-opt fnt-nme \xxfamily
4452 \let\bbbl@tempe\bbbl@mapselect
4453 \let\bbbl@mapselect\relax
4454 \let\bbbl@temp@fam#4% eg, '\rmfamily', to be restored below
4455 \let#4\@empty % Make sure \renewfontfamily is valid
4456 \bbbl@exp{%
4457 \let\bbbl@temp@pfam\<\bbbl@stripslash#4\space>% eg, '\rmfamily '
4458 \<keys_if_exist:nnF>{fontspec-opentype}{Script/\bbbl@cl{sname}}}%
4459 {\newfontscript{\bbbl@cl{sname}}{\bbbl@cl{sotf}}}%
4460 \<keys_if_exist:nnF>{fontspec-opentype}{Language/\bbbl@cl{lname}}}%
4461 {\newfontlanguage{\bbbl@cl{lname}}{\bbbl@cl{lotf}}}%
4462 \renewfontfamily\#4%
4463 [\bbbl@cl{lsys},#2]{#3}% ie \bbbl@exp{...}{#3}
4464 \begingroup
4465 #4%
4466 \xdef#1{\f@family}% eg, \bbbl@rmdflt@lang{FreeSerif(0)}
4467 \endgroup
4468 \let#4\bbbl@temp@fam
4469 \bbbl@exp{\let\<\bbbl@stripslash#4\space>\bbbl@temp@pfam
4470 \let\bbbl@mapselect\bbbl@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.

```

4471 \def\bbbl@font@rst#1#2#3#4{%
4472 \bbbl@csarg\def{famrst@#4}{\bbbl@font@set{#1}#2#3}}

```

The default font families. They are eurocentric, but the list can be expanded easily with \babelfont.

```

4473 \def\bbbl@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 :-).

```

4474 \newcommand\babelFSstore[2][{%
4475 \bbbl@ifblank{#1}%
4476 {\bbbl@csarg\def{sname@#2}{Latin}}%
4477 {\bbbl@csarg\def{sname@#2}{#1}}%
4478 \bbbl@provide@dirs{#2}%
4479 \bbbl@csarg\ifnum{wdir@#2}>\z@
4480 \let\bbbl@beforeforeign\leavevmode
4481 \EnableBabelHook{babel-bidi}%
4482 \fi
4483 \bbbl@foreach{#2}{%
4484 \bbbl@FSstore{##1}{rm}\rmdefault\bbbl@save@rmdefault
4485 \bbbl@FSstore{##1}{sf}\sfdefault\bbbl@save@sfdefault
4486 \bbbl@FSstore{##1}{tt}\ttdefault\bbbl@save@ttdefault}}
4487 \def\bbbl@FSstore#1#2#3#4{%
4488 \bbbl@csarg\edef{#2default#1}{#3}%
4489 \expandafter\addto\csname extras#1\endcsname{%
4490 \let#4#3%
4491 \ifx#3\f@family
4492 \edef#3{\csname bbl@#2default#1\endcsname}%
4493 \fontfamily{#3}\selectfont
4494 \else
4495 \edef#3{\csname bbl@#2default#1\endcsname}%
4496 \fi}%

```

```

4497 \expandafter\addto\csname noextras#1\endcsname{%
4498   \ifx#3\f@family
4499     \fontfamily{#4}\selectfont
4500     \fi
4501     \let#3#4}}
4502 \let\bbbl@langfeatures\@empty
4503 \def\babelFSfeatures{% make sure \fontspec is redefined once
4504   \let\bbbl@ori@fontspec\fontspec
4505   \renewcommand\fontspec[1][{}]{%
4506     \bbbl@ori@fontspec[\bbbl@langfeatures##1]}
4507   \let\babelFSfeatures\bbbl@FSfeatures
4508   \babelFSfeatures}
4509 \def\bbbl@FSfeatures#1#2{%
4510   \expandafter\addto\csname extras#1\endcsname{%
4511     \babel@save\bbbl@langfeatures
4512     \edef\bbbl@langfeatures{#2,}}
4513 <</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.

```

4514 <<(*Footnote changes)>> ≡
4515 \bbbl@trace{Bidi footnotes}
4516 \ifnum\bbbl@bidimode>\z@
4517   \def\bbbl@footnote#1#2#3{%
4518     \@ifnextchar[%
4519       {\bbbl@footnote@o{#1}{#2}{#3}}%
4520       {\bbbl@footnote@x{#1}{#2}{#3}}}
4521   \long\def\bbbl@footnote@x#1#2#3#4{%
4522     \bgroup
4523     \select@language@x{\bbbl@main@language}%
4524     \bbbl@fn@footnote{#2#1{\ignorespaces#4}#3}%
4525     \egroup}
4526   \long\def\bbbl@footnote@o#1#2#3[#4]#5{%
4527     \bgroup
4528     \select@language@x{\bbbl@main@language}%
4529     \bbbl@fn@footnote[#4]{#2#1{\ignorespaces#5}#3}%
4530     \egroup}
4531   \def\bbbl@footnotetext#1#2#3{%
4532     \@ifnextchar[%
4533       {\bbbl@footnotetext@o{#1}{#2}{#3}}%
4534       {\bbbl@footnotetext@x{#1}{#2}{#3}}}
4535   \long\def\bbbl@footnotetext@x#1#2#3#4{%
4536     \bgroup
4537     \select@language@x{\bbbl@main@language}%
4538     \bbbl@fn@footnotetext{#2#1{\ignorespaces#4}#3}%
4539     \egroup}
4540   \long\def\bbbl@footnotetext@o#1#2#3[#4]#5{%
4541     \bgroup
4542     \select@language@x{\bbbl@main@language}%
4543     \bbbl@fn@footnotetext[#4]{#2#1{\ignorespaces#5}#3}%
4544     \egroup}
4545   \def\BabelFootnote#1#2#3#4{%
4546     \ifx\bbbl@fn@footnote\@undefined
4547       \let\bbbl@fn@footnote\footnote

```

```

4548 \fi
4549 \ifx\bbbl@fn@footnotetext\undefined
4550 \let\bbbl@fn@footnotetext\footnotetext
4551 \fi
4552 \bbbl@ifblank{#2}%
4553 {\def#1{\bbbl@footnote{\@firstofone}{#3}{#4}}
4554 \namedef{\bbbl@stripslash#1text}%
4555 {\bbbl@footnotetext{\@firstofone}{#3}{#4}}}%
4556 {\def#1{\bbbl@exp{\bbbl@footnote{\foreignlanguage{#2}}{#3}{#4}}%
4557 \namedef{\bbbl@stripslash#1text}%
4558 {\bbbl@exp{\bbbl@footnotetext{\foreignlanguage{#2}}{#3}{#4}}}}
4559 \fi
4560 <</Footnote changes>>

```

Now, the code.

```

4561 (*xetex)
4562 \def\BabelStringsDefault{unicode}
4563 \let\xebbl@stop\relax
4564 \AddBabelHook{xetex}{encodedcommands}{%
4565 \def\bbbl@tempa{#1}%
4566 \ifx\bbbl@tempa\@empty
4567 \XeTeXinputencoding"bytes"%
4568 \else
4569 \XeTeXinputencoding"#1"%
4570 \fi
4571 \def\xebbl@stop{\XeTeXinputencoding"utf8"}}
4572 \AddBabelHook{xetex}{stopcommands}{%
4573 \xebbl@stop
4574 \let\xebbl@stop\relax}
4575 \def\bbbl@intraspace#1 #2 #3@@{%
4576 \bbbl@csarg\gdef{\xeisp@{language}}%
4577 {\XeTeXlinebreakskip #1em plus #2em minus #3em\relax}}
4578 \def\bbbl@intrapenalty#1@@{%
4579 \bbbl@csarg\gdef{\xeipn@{language}}%
4580 {\XeTeXlinebreakpenalty #1\relax}}
4581 \def\bbbl@provide@intraspace{%
4582 \bbbl@xin@{/s}{\bbbl@cl{lnbrk}}%
4583 \ifin@else\bbbl@xin@{/c}{\bbbl@cl{lnbrk}}\fi
4584 \ifin@
4585 \bbbl@ifunset{\bbbl@intsp@{language}}{%
4586 {\expandafter\ifx\csname \bbbl@intsp@{language}\endcsname\@empty\else
4587 \ifx\bbbl@KVP@intraspace\@nil
4588 \bbbl@exp{%
4589 \bbbl@intraspace\bbbl@cl{intsp}\@}%
4590 \fi
4591 \ifx\bbbl@KVP@intrapenalty\@nil
4592 \bbbl@intrapenalty0@@
4593 \fi
4594 \fi
4595 \ifx\bbbl@KVP@intraspace\@nil\else % We may override the ini
4596 \expandafter\bbbl@intraspace\bbbl@KVP@intraspace@@
4597 \fi
4598 \ifx\bbbl@KVP@intrapenalty\@nil\else
4599 \expandafter\bbbl@intrapenalty\bbbl@KVP@intrapenalty@@
4600 \fi
4601 \bbbl@exp{%
4602 % TODO. Execute only once (but redundant):
4603 \bbbl@add{<extras>{language}}%
4604 \XeTeXlinebreaklocale "\bbbl@cl{tbc}}}%

```

```

4605 \<bbl@xeisp@\languagenam>%
4606 \<bbl@xeipn@\languagenam>}%
4607 \\\bbl@toglobal\<extras\languagenam>%
4608 \\\bbl@add\<noextras\languagenam>{%
4609 \XeTeXlinebreaklocale "en"%
4610 \\\bbl@toglobal\<noextras\languagenam>}%
4611 \ifx\bbl@ispace\undefined
4612 \gdef\bbl@ispace{\bbl@cl{xeisp}}%
4613 \ifx\AtBeginDocument\@notprerr
4614 \expandafter\@secondoftwo % to execute right now
4615 \fi
4616 \AtBeginDocument{\bbl@patchfont{\bbl@ispace}}%
4617 \fi}%
4618 \fi}
4619 \ifx\DisableBabelHook\undefined\endinput\fi
4620 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
4621 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@cckstdfont}
4622 \DisableBabelHook{babel-fontspec}
4623 <<Font selection>>
4624 \input txtbabel.def
4625 </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>TEX</sub> and xet<sub>EX</sub>.

```

4626 <*texxet>
4627 \providecommand\bbl@provide@intraspace{}
4628 \bbl@trace{Redefinitions for bidi layout}
4629 \def\bbl@sspre@caption{%
4630 \bbl@exp{\everyhbox{\bbl@texmdir\bbl@cs{wdir@\bbl@main@language}}}}
4631 \ifx\bbl@opt@layout\@nnil\endinput\fi % No layout
4632 \def\bbl@startskip{\ifcase\bbl@thepardir\leftskip\else\rightskip\fi}
4633 \def\bbl@endskip{\ifcase\bbl@thepardir\rightskip\else\leftskip\fi}
4634 \ifx\bbl@beforeforeign\leavevmode % A poor test for bidi=
4635 \def\@hangfrom#1{%
4636 \setbox\@tempboxa\hbox{#1}%
4637 \hangindent\ifcase\bbl@thepardir\wd\@tempboxa\else-\wd\@tempboxa\fi
4638 \noindent\box\@tempboxa}
4639 \def\raggedright{%
4640 \let\@centercr
4641 \bbl@startskip\z@skip
4642 \@rightskip\@flushglue
4643 \bbl@endskip\@rightskip
4644 \parindent\z@
4645 \parfillskip\bbl@startskip}
4646 \def\raggedleft{%
4647 \let\@centercr
4648 \bbl@startskip\@flushglue
4649 \bbl@endskip\z@skip
4650 \parindent\z@
4651 \parfillskip\bbl@endskip}
4652 \fi

```

```

4653 \IfBabelLayout{lists}
4654   {\bbl@sreplace\list
4655     {\@totalleftmargin\leftmargin}{\@totalleftmargin\bbl@listleftmargin}%
4656     \def\bbl@listleftmargin{%
4657       \ifcase\bbl@thepardir\leftmargin\else\rightmargin\fi}%
4658     \ifcase\bbl@engine
4659       \def\labelenumii{}\theenumii{}\pdfTeX doesn't reverse ()
4660       \def\p@enumiii{\p@enumii}\theenumii{}\fi
4661     \fi
4662     \bbl@sreplace\@verbatim
4663       {\leftskip\@totalleftmargin}%
4664       {\bbl@startskip\textwidth
4665         \advance\bbl@startskip-\linewidth}%
4666     \bbl@sreplace\@verbatim
4667       {\rightskip\z@skip}%
4668       {\bbl@endskip\z@skip}}%
4669   {}
4670 \IfBabelLayout{contents}
4671   {\bbl@sreplace\@dottedtocline{\leftskip}{\bbl@startskip}%
4672     \bbl@sreplace\@dottedtocline{\rightskip}{\bbl@endskip}}
4673   {}
4674 \IfBabelLayout{columns}
4675   {\bbl@sreplace\@outputdblcol{\hb@xt@\textwidth}{\bbl@outputbox}%
4676     \def\bbl@outputbox#1{%
4677       \hb@xt@\textwidth{%
4678         \hskip\columnwidth
4679         \hfil
4680         {\normalcolor\vrule \@width\columnseprule}%
4681         \hfil
4682         \hb@xt@\columnwidth{\box\@leftcolumn \hss}%
4683         \hskip-\textwidth
4684         \hb@xt@\columnwidth{\box\@outputbox \hss}%
4685         \hskip\columnsep
4686         \hskip\columnwidth}}}%
4687   {}
4688 <<Footnote changes>>
4689 \IfBabelLayout{footnotes}%
4690   {\BabelFootnote\footnote\language\{}}%
4691   {\BabelFootnote\localfootnote\language\{}}%
4692   {\BabelFootnote\mainfootnote\{}}%
4693   {}

```

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.

```

4694 \IfBabelLayout{counters}%
4695   {\let\bbl@latinarabic=\@arabic
4696     \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
4697     \let\bbl@asciroman=\@roman
4698     \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
4699     \let\bbl@asciiRoman=\@Roman
4700     \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}%
4701 </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`).

```

4702 (*luatex)
4703 \ifx\AddBabelHook\undefined % When plain.def, babel.sty starts
4704 \bbl@trace{Read language.dat}
4705 \ifx\bbl@readstream\undefined
4706 \csname newread\endcsname\bbl@readstream
4707 \fi
4708 \begingroup
4709 \toks@{}
4710 \count@ \z@ % 0=start, 1=0th, 2=normal
4711 \def\bbl@process@line#1#2 #3 #4 {%
4712   \ifx=#1%
4713     \bbl@process@synonym{#2}%
4714   \else
4715     \bbl@process@language{#1#2}{#3}{#4}%
4716   \fi
4717   \ignorespaces}
4718 \def\bbl@manylang{%
4719   \ifnum\bbl@last>\@ne
4720     \bbl@info{Non-standard hyphenation setup}%
4721   \fi
4722   \let\bbl@manylang\relax}
4723 \def\bbl@process@language#1#2#3{%
4724   \ifcase\count@
4725     \@ifundefined{zth@#1}{\count@\tw@}{\count@\@ne}%
4726   \or
4727     \count@\tw@
4728   \fi
4729   \ifnum\count@=\tw@
4730     \expandafter\addlanguage\csname l@#1\endcsname
4731     \language\allocationnumber

```

```

4732 \chardef\bbl@last\allocationnumber
4733 \bbl@manylang
4734 \let\bbl@elt\relax
4735 \xdef\bbl@languages{%
4736 \bbl@languages\bbl@elt{#1}{\the\language}{#2}{#3}}%
4737 \fi
4738 \the\toks@
4739 \toks@{}}
4740 \def\bbl@process@synonym@aux#1#2{%
4741 \global\expandafter\chardef\csname l@#1\endcsname#2\relax
4742 \let\bbl@elt\relax
4743 \xdef\bbl@languages{%
4744 \bbl@languages\bbl@elt{#1}{#2}{}}}%
4745 \def\bbl@process@synonym#1{%
4746 \ifcase\count@
4747 \toks@\expandafter{\the\toks@\relax\bbl@process@synonym{#1}}%
4748 \or
4749 \@ifundefined{zth@#1}{\bbl@process@synonym@aux{#1}{}}}%
4750 \else
4751 \bbl@process@synonym@aux{#1}{\the\bbl@last}%
4752 \fi}
4753 \ifx\bbl@languages\@undefined % Just a (sensible?) guess
4754 \chardef\l@english\z@
4755 \chardef\l@USenglish\z@
4756 \chardef\bbl@last\z@
4757 \global\@namedef{bbl@hyphendata@0}{\hyphen.tex}}
4758 \gdef\bbl@languages{%
4759 \bbl@elt{english}{0}{\hyphen.tex}}%
4760 \bbl@elt{USenglish}{0}{}}
4761 \else
4762 \global\let\bbl@languages@format\bbl@languages
4763 \def\bbl@elt#1#2#3#4{% Remove all except language 0
4764 \ifnum#2>\z@\else
4765 \noexpand\bbl@elt{#1}{#2}{#3}{#4}%
4766 \fi}%
4767 \xdef\bbl@languages{\bbl@languages}%
4768 \fi
4769 \def\bbl@elt#1#2#3#4{\@namedef{zth@#1}{}} % Define flags
4770 \bbl@languages
4771 \openin\bbl@readstream=language.dat
4772 \ifeof\bbl@readstream
4773 \bbl@warning{I couldn't find language.dat. No additional\\%
4774 patterns loaded. Reported}%
4775 \else
4776 \loop
4777 \endlinechar\m@ne
4778 \read\bbl@readstream to \bbl@line
4779 \endlinechar``^^M
4780 \if T\ifeof\bbl@readstream F\fi T\relax
4781 \ifx\bbl@line\@empty\else
4782 \edef\bbl@line{\bbl@line\space\space\space}%
4783 \expandafter\bbl@process@line\bbl@line\relax
4784 \fi
4785 \repeat
4786 \fi
4787 \endgroup
4788 \bbl@trace{Macros for reading patterns files}
4789 \def\bbl@get@enc#1:#2:#3\@@@{\def\bbl@hyph@enc{#2}}
4790 \ifx\babelcatcodetablenum\@undefined

```

```

4791 \ifx\newcatcodetable\@undefined
4792 \def\babelcatcodetablenum{5211}
4793 \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4794 \else
4795 \newcatcodetable\babelcatcodetablenum
4796 \newcatcodetable\bbl@pattcodes
4797 \fi
4798 \else
4799 \def\bbl@pattcodes{\numexpr\babelcatcodetablenum+1\relax}
4800 \fi
4801 \def\bbl@luapatterns#1#2{%
4802 \bbl@get@enc#1::@@@
4803 \setbox\z@\hbox\bgroup
4804 \begingroup
4805 \savecatcodetable\babelcatcodetablenum\relax
4806 \initcatcodetable\bbl@pattcodes\relax
4807 \catcodetable\bbl@pattcodes\relax
4808 \catcode`\#=6 \catcode`\$=3 \catcode`\&=4 \catcode`\^=7
4809 \catcode`\_ =8 \catcode`\{=1 \catcode`\}=2 \catcode`\~=13
4810 \catcode`\@=11 \catcode`\^^I=10 \catcode`\^^J=12
4811 \catcode`\<=12 \catcode`\>=12 \catcode`\*=12 \catcode`\.=12
4812 \catcode`\-=12 \catcode`\/=12 \catcode`\[=12 \catcode`\]=12
4813 \catcode`\`=12 \catcode`\'=12 \catcode`\`=12
4814 \input #1\relax
4815 \catcodetable\babelcatcodetablenum\relax
4816 \endgroup
4817 \def\bbl@tempa{#2}%
4818 \ifx\bbl@tempa\@empty\else
4819 \input #2\relax
4820 \fi
4821 \egroup}%
4822 \def\bbl@patterns@lua#1{%
4823 \language=\expandafter\ifx\csname l@#1:\f@encoding\endcsname\relax
4824 \csname l@#1\endcsname
4825 \edef\bbl@tempa{#1}%
4826 \else
4827 \csname l@#1:\f@encoding\endcsname
4828 \edef\bbl@tempa{#1:\f@encoding}%
4829 \fi\relax
4830 \@namedef{lu@texhyphen@loaded@the\language}{}% Temp
4831 \@ifundefined{bbl@hyphendata@the\language}%
4832 {\def\bbl@elt##1##2##3##4{%
4833 \ifnum##2=\csname l@bbl@tempa\endcsname % #2=spanish, dutch:OT1...
4834 \def\bbl@tempb{##3}%
4835 \ifx\bbl@tempb\@empty\else % if not a synonymous
4836 \def\bbl@tempc{##3}{##4}%
4837 \fi
4838 \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
4839 \fi}%
4840 \bbl@languages
4841 \@ifundefined{bbl@hyphendata@the\language}%
4842 {\bbl@info{No hyphenation patterns were set for\%
4843 language '\bbl@tempa'. Reported}}%
4844 {\expandafter\expandafter\expandafter\bbl@luapatterns
4845 \csname bbl@hyphendata@the\language\endcsname}}}%
4846 \endinput\fi
4847 % Here ends \ifx\AddBabelHook\@undefined
4848 % A few lines are only read by hyphen.cfg
4849 \ifx\DisableBabelHook\@undefined

```

```

4850 \AddBabelHook{luatex}{everylanguage}{%
4851   \def\process@language##1##2##3{%
4852     \def\process@line####1####2 ####3 ####4 {}}
4853 \AddBabelHook{luatex}{loadpatterns}{%
4854   \input #1\relax
4855   \expandafter\gdef\csname bbl@hyphendata@\the\language\endcsname
4856     {{#1}{}}}
4857 \AddBabelHook{luatex}{loadexceptions}{%
4858   \input #1\relax
4859   \def\bbl@tempb##1##2{{##1}{#1}}%
4860   \expandafter\xdef\csname bbl@hyphendata@\the\language\endcsname
4861     {\expandafter\expandafter\expandafter\bbl@tempb
4862       \csname bbl@hyphendata@\the\language\endcsname}}
4863 \endinput\fi
4864 % Here stops reading code for hyphen.cfg
4865 % The following is read the 2nd time it's loaded
4866 \begingroup % TODO - to a lua file
4867 \catcode`\%=12
4868 \catcode`\'=12
4869 \catcode`\%=12
4870 \catcode`\:=12
4871 \directlua{
4872   Babel = Babel or {}
4873   function Babel.bytes(line)
4874     return line:gsub(".",
4875       function (chr) return unicode.utf8.char(string.byte(chr)) end)
4876   end
4877   function Babel.begin_process_input()
4878     if luatexbase and luatexbase.add_to_callback then
4879       luatexbase.add_to_callback('process_input_buffer',
4880         Babel.bytes, 'Babel.bytes')
4881     else
4882       Babel.callback = callback.find('process_input_buffer')
4883       callback.register('process_input_buffer', Babel.bytes)
4884     end
4885   end
4886   function Babel.end_process_input ()
4887     if luatexbase and luatexbase.remove_from_callback then
4888       luatexbase.remove_from_callback('process_input_buffer', 'Babel.bytes')
4889     else
4890       callback.register('process_input_buffer', Babel.callback)
4891     end
4892   end
4893   function Babel.addpatterns(pp, lg)
4894     local lg = lang.new(lg)
4895     local pats = lang.patterns(lg) or ''
4896     lang.clear_patterns(lg)
4897     for p in pp:gmatch('[^%s]+') do
4898       ss = ''
4899       for i in string.utfcharacters(p:gsub('%d', '')) do
4900         ss = ss .. '%d?' .. i
4901       end
4902       ss = ss:gsub('^%%d%?%', '%%.') .. '%d?'
4903       ss = ss:gsub('%.%%d%?$', '%%.')
4904       pats, n = pats:gsub('%s' .. ss .. '%s', ' ' .. p .. ' ')
4905       if n == 0 then
4906         tex.sprint(
4907           [[\string\csname\space bbl@info\endcsname{New pattern: }
4908           .. p .. [{}]])

```

```

4909     pats = pats .. ' ' .. p
4910   else
4911     tex.sprint(
4912       [[\string\csname\space bbl@info\endcsname{Renew pattern: }]]
4913       .. p .. [[]])
4914   end
4915 end
4916 lang.patterns(lg, pats)
4917 end
4918 }
4919 \endgroup
4920 \ifx\newattribute\@undefined\else
4921   \newattribute\bbl@attr@locale
4922   \directlua{ Babel.attr_locale = luatexbase.registernumber'bbl@attr@locale' }
4923   \AddBabelHook{luatex}{beforeextras}{%
4924     \setattribute\bbl@attr@locale\localeid}
4925 \fi
4926 \def\BabelStringsDefault{unicode}
4927 \let\luabbl@stop\relax
4928 \AddBabelHook{luatex}{encodedcommands}{%
4929   \def\bbl@tempa{utf8}\def\bbl@tempb{#1}%
4930   \ifx\bbl@tempa\bbl@tempb\else
4931     \directlua{Babel.begin_process_input()}%
4932     \def\luabbl@stop{%
4933       \directlua{Babel.end_process_input()}}%
4934   \fi}%
4935 \AddBabelHook{luatex}{stopcommands}{%
4936   \luabbl@stop
4937   \let\luabbl@stop\relax}
4938 \AddBabelHook{luatex}{patterns}{%
4939   \@ifundefined{bbl@hyphendata@the\language}%
4940   {\def\bbl@elt##1##2##3##4{%
4941     \ifnum##2=\csname l@##2\endcsname % #2=spanish, dutch:OT1...
4942     \def\bbl@tempb{##3}%
4943     \ifx\bbl@tempb\@empty\else % if not a synonymous
4944       \def\bbl@tempc{##3}{##4}}%
4945     \fi
4946     \bbl@csarg\xdef{hyphendata@##2}{\bbl@tempc}%
4947   \fi}%
4948   \bbl@languages
4949   \@ifundefined{bbl@hyphendata@the\language}%
4950   {\bbl@info{No hyphenation patterns were set for\%
4951     language '#2'. Reported}}%
4952   {\expandafter\expandafter\expandafter\bbl@luapatterns
4953     \csname bbl@hyphendata@the\language\endcsname}}}%
4954   \@ifundefined{bbl@patterns@}{}%
4955   \begingroup
4956     \bbl@xin@{,\number\language,}{,\bbl@pttnlist}%
4957     \ifin@else
4958       \ifx\bbl@patterns@\@empty\else
4959         \directlua{ Babel.addpatterns(
4960           [[\bbl@patterns@]], \number\language) }%
4961       \fi
4962       \@ifundefined{bbl@patterns@#1}%
4963       \@empty
4964       {\directlua{ Babel.addpatterns(
4965         [[\space\csname bbl@patterns@#1\endcsname]],
4966         \number\language) }}%
4967       \xdef\bbl@pttnlist{\bbl@pttnlist\number\language,}%

```

```

4968     \fi
4969   \endgroup}%
4970 \bbl@exp{%
4971   \bbl@ifunset{bbl@prehc\language}{}%
4972   {\bbl@i fblank{\bbl@cs{prehc\language}}}%
4973   {\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.

```

4974 \@onlypreamble\babelpatterns
4975 \AtEndOfPackage{%
4976   \newcommand\babelpatterns[2][\@empty]{%
4977     \ifx\bbl@patterns@relax
4978       \let\bbl@patterns@\@empty
4979     \fi
4980     \ifx\bbl@pttnlist\@empty\else
4981       \bbl@warning{%
4982         You must not intermingle \string\selectlanguage\space and\%
4983         \string\babelpatterns\space or some patterns will not\%
4984         be taken into account. Reported}%
4985     \fi
4986     \ifx\@empty#1%
4987       \protected@edef\bbl@patterns@{\bbl@patterns@\space#2}%
4988     \else
4989       \edef\bbl@tempb{\zap@space#1 \@empty}%
4990       \bbl@for\bbl@tempa\bbl@tempb{%
4991         \bbl@fixname\bbl@tempa
4992         \bbl@iflanguage\bbl@tempa{%
4993           \bbl@csarg\protected@edef{patterns@\bbl@tempa}{%
4994             \ifundefined{bbl@patterns@\bbl@tempa}%
4995               \@empty
4996               {\csname bbl@patterns@\bbl@tempa\endcsname\space}%
4997             #2}}%
4998       \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.

```

4999% TODO - to a lua file
5000 \directlua{
5001   Babel = Babel or {}
5002   Babel.linebreaking = Babel.linebreaking or {}
5003   Babel.linebreaking.before = {}
5004   Babel.linebreaking.after = {}
5005   Babel.locale = {} % Free to use, indexed by \localeid
5006   function Babel.linebreaking.add_before(func)
5007     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5008     table.insert(Babel.linebreaking.before, func)
5009   end
5010   function Babel.linebreaking.add_after(func)
5011     tex.print([[noexpand\csname bbl@luahyphenate\endcsname]])
5012     table.insert(Babel.linebreaking.after, func)
5013   end
5014 }

```

```

5015 \def\bbl@intraspace#1 #2 #3\@@{%
5016 \directlua{
5017   Babel = Babel or {}
5018   Babel.intraspaces = Babel.intraspaces or {}
5019   Babel.intraspaces['\csname bbl@sbc@language\endcsname'] = %
5020     {b = #1, p = #2, m = #3}
5021   Babel.locale_props[\the\localeid].intraspace = %
5022     {b = #1, p = #2, m = #3}
5023 }}
5024 \def\bbl@intrapenalty#1\@@{%
5025 \directlua{
5026   Babel = Babel or {}
5027   Babel.intrapenalties = Babel.intrapenalties or {}
5028   Babel.intrapenalties['\csname bbl@sbc@language\endcsname'] = #1
5029   Babel.locale_props[\the\localeid].intrapenalty = #1
5030 }}
5031 \begingroup
5032 \catcode`\%=12
5033 \catcode`\^=14
5034 \catcode`\'=12
5035 \catcode`\~=12
5036 \gdef\bbl@seaintraspace{^
5037 \let\bbl@seaintraspace\relax
5038 \directlua{
5039   Babel = Babel or {}
5040   Babel.sea_enabled = true
5041   Babel.sea_ranges = Babel.sea_ranges or {}
5042   function Babel.set_chranges (script, chrng)
5043     local c = 0
5044     for s, e in string.gmatch(chrng..' ', '(.-%.%.(-)%s') do
5045       Babel.sea_ranges[script..c]={tonumber(s,16), tonumber(e,16)}
5046       c = c + 1
5047     end
5048   end
5049   function Babel.sea_disc_to_space (head)
5050     local sea_ranges = Babel.sea_ranges
5051     local last_char = nil
5052     local quad = 655360 ^% 10 pt = 655360 = 10 * 65536
5053     for item in node.traverse(head) do
5054       local i = item.id
5055       if i == node.id'glyph' then
5056         last_char = item
5057       elseif i == 7 and item.subtype == 3 and last_char
5058         and last_char.char > 0x0C99 then
5059         quad = font.getfont(last_char.font).size
5060         for lg, rg in pairs(sea_ranges) do
5061           if last_char.char > rg[1] and last_char.char < rg[2] then
5062             lg = lg:sub(1, 4) ^% Remove trailing number of, eg, Cyril1
5063             local intraspace = Babel.intraspaces[lg]
5064             local intrapenalty = Babel.intrapenalties[lg]
5065             local n
5066             if intrapenalty ~= 0 then
5067               n = node.new(14, 0) ^% penalty
5068               n.penalty = intrapenalty
5069               node.insert_before(head, item, n)
5070             end
5071             n = node.new(12, 13) ^% (glue, spaceskip)
5072             node.setglue(n, intraspace.b * quad,
5073               intraspace.p * quad,

```

```

5074             intraspace.m * quad)
5075             node.insert_before(head, item, n)
5076             node.remove(head, item)
5077         end
5078     end
5079 end
5080 end
5081 end
5082 }^^
5083 \bbl@luahyphenate}

```

### 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.

```

5084 \catcode`\%=14
5085 \gdef\bbl@cjk intraspace{%
5086   \let\bbl@cjk intraspace\relax
5087   \directlua{
5088     Babel = Babel or {}
5089     require('babel-data-cjk.lua')
5090     Babel.cjk_enabled = true
5091     function Babel.cjk_linebreak(head)
5092       local GLYPH = node.id'glyph'
5093       local last_char = nil
5094       local quad = 655360      % 10 pt = 655360 = 10 * 65536
5095       local last_class = nil
5096       local last_lang = nil
5097
5098       for item in node.traverse(head) do
5099         if item.id == GLYPH then
5100
5101           local lang = item.lang
5102
5103           local LOCALE = node.get_attribute(item,
5104             Babel.attr_locale)
5105           local props = Babel.locale_props[LOCALE]
5106
5107           local class = Babel.cjk_class[item.char].c
5108
5109           if props.cjk_quotes and props.cjk_quotes[item.char] then
5110             class = props.cjk_quotes[item.char]
5111           end
5112
5113           if class == 'cp' then class = 'cl' end % ]] as CL
5114           if class == 'id' then class = 'I' end
5115
5116           local br = 0
5117           if class and last_class and Babel.cjk_breaks[last_class][class] then
5118             br = Babel.cjk_breaks[last_class][class]
5119           end
5120
5121           if br == 1 and props.linebreak == 'c' and
5122             lang ~= \the\l@nohyphenation\space and

```



```

5123         last_lang ~= \the\l@nohyphenation then
5124         local intrapenalty = props.intrapenalty
5125         if intrapenalty ~= 0 then
5126             local n = node.new(14, 0)    % penalty
5127             n.penalty = intrapenalty
5128             node.insert_before(head, item, n)
5129         end
5130         local intraspace = props.intraspace
5131         local n = node.new(12, 13)    % (glue, spaceskip)
5132         node.setglue(n, intraspace.b * quad,
5133             intraspace.p * quad,
5134             intraspace.m * quad)
5135         node.insert_before(head, item, n)
5136     end
5137
5138     if font.getfont(item.font) then
5139         quad = font.getfont(item.font).size
5140     end
5141     last_class = class
5142     last_lang = lang
5143     else % if penalty, glue or anything else
5144         last_class = nil
5145     end
5146 end
5147 lang.hyphenate(head)
5148 end
5149 }%
5150 \bbl@luahyphenate}
5151 \gdef\bbl@luahyphenate{%
5152 \let\bbl@luahyphenate\relax
5153 \directlua{
5154     luatexbase.add_to_callback('hyphenate',
5155     function (head, tail)
5156         if Babel.linebreaking.before then
5157             for k, func in ipairs(Babel.linebreaking.before) do
5158                 func(head)
5159             end
5160         end
5161         if Babel.cjk_enabled then
5162             Babel.cjk_linebreak(head)
5163         end
5164         lang.hyphenate(head)
5165         if Babel.linebreaking.after then
5166             for k, func in ipairs(Babel.linebreaking.after) do
5167                 func(head)
5168             end
5169         end
5170         if Babel.sea_enabled then
5171             Babel.sea_disc_to_space(head)
5172         end
5173     end,
5174     'Babel.hyphenate')
5175 }
5176 }
5177 \endgroup
5178 \def\bbl@provide@intraspace{%
5179 \bbl@ifunset{bbl@intsp@language}{%
5180     {\expandafter\ifx\csname bbl@intsp@language\endcsname\@empty\else
5181         \bbl@xin@{c}{\bbl@cl{lnbrk}}}%

```

```

5182 \ifin@ % cjk
5183 \bbl@cjkintraspacespace
5184 \directlua{
5185     Babel = Babel or {}
5186     Babel.locale_props = Babel.locale_props or {}
5187     Babel.locale_props[\the\localeid].linebreak = 'c'
5188 }%
5189 \bbl@exp{\bbl@intraspacespace\bbl@cl{intsp}\bbl@cl{intsp}\bbl@cl{intsp}\bbl@cl{intsp}}%
5190 \ifx\bbl@KVP@intrapenalty\@nil
5191 \bbl@intrapenalty0\@
5192 \fi
5193 \else % sea
5194 \bbl@seaintraspacespace
5195 \bbl@exp{\bbl@intraspacespace\bbl@cl{intsp}\bbl@cl{intsp}\bbl@cl{intsp}\bbl@cl{intsp}}%
5196 \directlua{
5197     Babel = Babel or {}
5198     Babel.sea_ranges = Babel.sea_ranges or {}
5199     Babel.set_chranges('\bbl@cl{sbc}\bbl@cl{sbc}',
5200                       '\bbl@cl{chrng}\bbl@cl{chrng}')
5201 }%
5202 \ifx\bbl@KVP@intrapenalty\@nil
5203 \bbl@intrapenalty0\@
5204 \fi
5205 \fi
5206 \fi
5207 \ifx\bbl@KVP@intrapenalty\@nil\else
5208 \expandafter\bbl@intrapenalty\bbl@KVP@intrapenalty\@
5209 \fi}}

```

### 13.6 Arabic justification

```

5210 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5211 \def\bblar@chars{%
5212     0628,0629,062A,062B,062C,062D,062E,062F,0630,0631,0632,0633,%
5213     0634,0635,0636,0637,0638,0639,063A,063B,063C,063D,063E,063F,%
5214     0640,0641,0642,0643,0644,0645,0646,0647,0649}
5215 \def\bblar@elongated{%
5216     0626,0628,062A,062B,0633,0634,0635,0636,063B,%
5217     063C,063D,063E,063F,0641,0642,0643,0644,0646,%
5218     0649,064A}
5219 \begingroup
5220 \catcode\_ =11 \catcode\` =11
5221 \gdef\bblar@nofswarn{\gdef\msg_warning:nx##1##2##3{}}
5222 \endgroup
5223 \gdef\bbl@arabicjust{%
5224     \let\bbl@arabicjust\relax
5225     \newattribute\bblar@kashida
5226     \directlua{ Babel.attr_kashida = luatexbase.registernumber'bblar@kashida' }%
5227     \bblar@kashida=\z@
5228     \bbl@patchfont{\bbl@parsejalt}}%
5229 \directlua{
5230     Babel.arabic.elong_map = Babel.arabic.elong_map or {}
5231     Babel.arabic.elong_map[\the\localeid] = {}
5232     luatexbase.add_to_callback('post_linebreak_filter',
5233     Babel.arabic.justify, 'Babel.arabic.justify')
5234     luatexbase.add_to_callback('hpack_filter',
5235     Babel.arabic.justify_hbox, 'Babel.arabic.justify_hbox')
5236 }}%
5237 % Save both node lists to make replacement. TODO. Save also widths to

```

```

5238% make computations
5239 \def\bblar@fetchjalt#1#2#3#4{%
5240   \bbl@exp{\bbl@foreach{#1}}{%
5241     \bbl@ifunset\bblar@JE@##1}%
5242     {\setbox\z@\hbox{^^^200d\char"##1#2}}%
5243     {\setbox\z@\hbox{^^^200d\char"@nameuse\bblar@JE@##1#2}}%
5244   \directlua{%
5245     local last = nil
5246     for item in node.traverse(tex.box[0].head) do
5247       if item.id == node.id'glyph' and item.char > 0x600 and
5248         not (item.char == 0x200D) then
5249         last = item
5250       end
5251     end
5252     Babel.arabic.#3['##1#4'] = last.char
5253   }}
5254% Brute force. No rules at all, yet. The ideal: look at jalt table. And
5255% perhaps other tables (falt?, csw?). What about kaf? And diacritic
5256% positioning?
5257 \gdef\bbl@parsejalt{%
5258   \ifx\addfontfeature\undefined\else
5259     \bbl@xin@{/e}{/\bbl@cl{lbrk}}%
5260     \ifin@
5261       \directlua{%
5262         if Babel.arabic.elong_map[\the\localeid][\fontid\font] == nil then
5263           Babel.arabic.elong_map[\the\localeid][\fontid\font] = {}
5264           tex.print([[string\csname\space\bbl@parsejalti\endcsname]])
5265         end
5266       }%
5267     \fi
5268   \fi}
5269 \gdef\bbl@parsejalti{%
5270   \begingroup
5271     \let\bbl@parsejalt\relax % To avoid infinite loop
5272     \edef\bbl@tempb{\fontid\font}%
5273     \bblar@nofswarn
5274     \bblar@fetchjalt\bblar@elongated{}{from}{}%
5275     \bblar@fetchjalt\bblar@chars{^^^064a}{from}{a}% Alef maksura
5276     \bblar@fetchjalt\bblar@chars{^^^0649}{from}{y}% Yeh
5277     \addfontfeature{RawFeature+=jalt}%
5278     % \namedef\bblar@JE@0643{06AA}% todo: catch medial kaf
5279     \bblar@fetchjalt\bblar@elongated{}{dest}{}%
5280     \bblar@fetchjalt\bblar@chars{^^^064a}{dest}{a}%
5281     \bblar@fetchjalt\bblar@chars{^^^0649}{dest}{y}%
5282     \directlua{%
5283       for k, v in pairs(Babel.arabic.from) do
5284         if Babel.arabic.dest[k] and
5285           not (Babel.arabic.from[k] == Babel.arabic.dest[k]) then
5286           Babel.arabic.elong_map[\the\localeid][\bbl@tempb]
5287             [Babel.arabic.from[k]] = Babel.arabic.dest[k]
5288         end
5289       end
5290     }%
5291   \endgroup}
5292%
5293 \begingroup
5294 \catcode`#=11
5295 \catcode`~ =11
5296 \directlua{

```

```

5297
5298 Babel.arabic = Babel.arabic or {}
5299 Babel.arabic.from = {}
5300 Babel.arabic.dest = {}
5301 Babel.arabic.justify_factor = 0.95
5302 Babel.arabic.justify_enabled = true
5303
5304 function Babel.arabic.justify(head)
5305   if not Babel.arabic.justify_enabled then return head end
5306   for line in node.traverse_id(node.id'hlist', head) do
5307     Babel.arabic.justify_hlist(head, line)
5308   end
5309   return head
5310 end
5311
5312 function Babel.arabic.justify_hbox(head, gc, size, pack)
5313   local has_inf = false
5314   if Babel.arabic.justify_enabled and pack == 'exactly' then
5315     for n in node.traverse_id(12, head) do
5316       if n.stretch_order > 0 then has_inf = true end
5317     end
5318     if not has_inf then
5319       Babel.arabic.justify_hlist(head, nil, gc, size, pack)
5320     end
5321   end
5322   return head
5323 end
5324
5325 function Babel.arabic.justify_hlist(head, line, gc, size, pack)
5326   local d, new
5327   local k_list, k_item, pos_inline
5328   local width, width_new, full, k_curr, wt_pos, goal, shift
5329   local subst_done = false
5330   local elong_map = Babel.arabic.elong_map
5331   local last_line
5332   local GLYPH = node.id'glyph'
5333   local KASHIDA = Babel.attr_kashida
5334   local LOCALE = Babel.attr_locale
5335
5336   if line == nil then
5337     line = {}
5338     line.glue_sign = 1
5339     line.glue_order = 0
5340     line.head = head
5341     line.shift = 0
5342     line.width = size
5343   end
5344
5345   % Exclude last line. todo. But-- it discards one-word lines, too!
5346   % ? Look for glue = 12:15
5347   if (line.glue_sign == 1 and line.glue_order == 0) then
5348     elongs = {} % Stores elongated candidates of each line
5349     k_list = {} % And all letters with kashida
5350     pos_inline = 0 % Not yet used
5351
5352     for n in node.traverse_id(GLYPH, line.head) do
5353       pos_inline = pos_inline + 1 % To find where it is. Not used.
5354     end
5355     % Elongated glyphs

```

```

5356     if elong_map then
5357         local locale = node.get_attribute(n, LOCALE)
5358         if elong_map[locale] and elong_map[locale][n.font] and
5359             elong_map[locale][n.font][n.char] then
5360             table.insert(elongs, {node = n, locale = locale} )
5361             node.set_attribute(n.prev, KASHIDA, 0)
5362         end
5363     end
5364
5365     % Tatwil
5366     if Babel.kashida_wts then
5367         local k_wt = node.get_attribute(n, KASHIDA)
5368         if k_wt > 0 then % todo. parameter for multi inserts
5369             table.insert(k_list, {node = n, weight = k_wt, pos = pos_inline})
5370         end
5371     end
5372
5373     end % of node.traverse_id
5374
5375     if #elongs == 0 and #k_list == 0 then goto next_line end
5376     full = line.width
5377     shift = line.shift
5378     goal = full * Babel.arabic.justify_factor % A bit crude
5379     width = node.dimensions(line.head) % The 'natural' width
5380
5381     % == Elongated ==
5382     % Original idea taken from 'chickenize'
5383     while (#elongs > 0 and width < goal) do
5384         subst_done = true
5385         local x = #elongs
5386         local curr = elongs[x].node
5387         local oldchar = curr.char
5388         curr.char = elong_map[elongs[x].locale][curr.font][curr.char]
5389         width = node.dimensions(line.head) % Check if the line is too wide
5390         % Substitute back if the line would be too wide and break:
5391         if width > goal then
5392             curr.char = oldchar
5393             break
5394         end
5395         % If continue, pop the just substituted node from the list:
5396         table.remove(elongs, x)
5397     end
5398
5399     % == Tatwil ==
5400     if #k_list == 0 then goto next_line end
5401
5402     width = node.dimensions(line.head) % The 'natural' width
5403     k_curr = #k_list
5404     wt_pos = 1
5405
5406     while width < goal do
5407         subst_done = true
5408         k_item = k_list[k_curr].node
5409         if k_list[k_curr].weight == Babel.kashida_wts[wt_pos] then
5410             d = node.copy(k_item)
5411             d.char = 0x0640
5412             line.head, new = node.insert_after(line.head, k_item, d)
5413             width_new = node.dimensions(line.head)
5414             if width > goal or width == width_new then

```

```

5415         node.remove(line.head, new) % Better compute before
5416         break
5417     end
5418     width = width_new
5419 end
5420 if k_curr == 1 then
5421     k_curr = #k_list
5422     wt_pos = (wt_pos >= table.getn(Babel.kashida_wts)) and 1 or wt_pos+1
5423 else
5424     k_curr = k_curr - 1
5425 end
5426 end
5427
5428 ::next_line::
5429
5430 % Must take into account marks and ins, see luatex manual.
5431 % Have to be executed only if there are changes. Investigate
5432 % what's going on exactly.
5433 if subst_done and not gc then
5434     d = node.hpack(line.head, full, 'exactly')
5435     d.shift = shift
5436     node.insert_before(head, line, d)
5437     node.remove(head, line)
5438 end
5439 end % if process line
5440 end
5441 }
5442 \endgroup
5443 \fi\fi % Arabic just block

```

### 13.7 Common stuff

```

5444 \AddBabelHook{babel-fontspec}{afterextras}{\bbl@switchfont}
5445 \AddBabelHook{babel-fontspec}{beforestart}{\bbl@ckeckstdfonts}
5446 \DisableBabelHook{babel-fontspec}
5447 <<Font selection>>

```

### 13.8 Automatic fonts and ids switching

After defining the blocks for a number of scripts (must be extended and very likely fine tuned), we define a short function which just traverse the node list to carry out the replacements. The table `loc_to_scr` gets the locale form a script range (note the locale is the key, and that there is an intermediate table built on the fly for optimization). This locale is then used to get the `\language` and the `\localeid` as stored in `locale_props`, as well as the font (as requested). In the latter table a key starting with `/` maps the font from the global one (the key) to the local one (the value). Maths are skipped and discretionaries are handled in a special way.

```

5448 % TODO - to a lua file
5449 \directlua{
5450 Babel.script_blocks = {
5451   ['dflt'] = {},
5452   ['Arab'] = {{0x0600, 0x06FF}, {0x08A0, 0x08FF}, {0x0750, 0x077F},
5453               {0xFE70, 0xFEFF}, {0xFB50, 0xFDFF}, {0x1EE00, 0x1EEFF}},
5454   ['Armn'] = {{0x0530, 0x058F}},
5455   ['Beng'] = {{0x0980, 0x09FF}},
5456   ['Cher'] = {{0x13A0, 0x13FF}, {0xAB70, 0ABBF}},
5457   ['Copt'] = {{0x03E2, 0x03EF}, {0x2C80, 0x2CFF}, {0x102E0, 0x102FF}},
5458   ['Cyr1'] = {{0x0400, 0x04FF}, {0x0500, 0x052F}, {0x1C80, 0x1C8F},
5459               {0x2DE0, 0x2DFF}, {0xA640, 0xA69F}},
5460   ['Deva'] = {{0x0900, 0x097F}, {0xA8E0, 0xA8FF}},
5461   ['Ethi'] = {{0x1200, 0x137F}, {0x1380, 0x139F}, {0x2D80, 0x2DDF}},

```

```

5462         {0xAB00, 0xAB2F}},
5463 ['Geor'] = {{0x10A0, 0x10FF}, {0x2D00, 0x2D2F}},
5464 % Don't follow strictly Unicode, which places some Coptic letters in
5465 % the 'Greek and Coptic' block
5466 ['Grek'] = {{0x0370, 0x03E1}, {0x03F0, 0x03FF}, {0x1F00, 0x1FFF}},
5467 ['Hans'] = {{0x2E80, 0x2EFF}, {0x3000, 0x303F}, {0x31C0, 0x31EF},
5468             {0x3300, 0x33FF}, {0x3400, 0x4DBF}, {0x4E00, 0x9FFF},
5469             {0xF900, 0xFAFF}, {0xFE30, 0xFE4F}, {0xFF00, 0xFFEF},
5470             {0x20000, 0x2A6DF}, {0x2A700, 0x2B73F},
5471             {0x2B740, 0x2B81F}, {0x2B820, 0x2CEAF},
5472             {0x2CEB0, 0x2EBEF}, {0x2F800, 0x2FA1F}},
5473 ['Hebr'] = {{0x0590, 0x05FF}},
5474 ['Jpan'] = {{0x3000, 0x303F}, {0x3040, 0x309F}, {0x30A0, 0x30FF},
5475             {0x4E00, 0x9FAF}, {0xFF00, 0xFFEF}},
5476 ['Khmr'] = {{0x1780, 0x17FF}, {0x19E0, 0x19FF}},
5477 ['Knda'] = {{0x0C80, 0x0CFF}},
5478 ['Kore'] = {{0x1100, 0x11FF}, {0x3000, 0x303F}, {0x3130, 0x318F},
5479             {0x4E00, 0x9FAF}, {0xA960, 0xA97F}, {0xAC00, 0xD7AF},
5480             {0xD7B0, 0xD7FF}, {0xFF00, 0xFFEF}},
5481 ['Lao'] = {{0x0E80, 0x0EFF}},
5482 ['Latn'] = {{0x0000, 0x007F}, {0x0080, 0x00FF}, {0x0100, 0x017F},
5483             {0x0180, 0x024F}, {0x1E00, 0x1EFF}, {0x2C60, 0x2C7F},
5484             {0xA720, 0xA7FF}, {0xAB30, 0xAB6F}},
5485 ['Mahj'] = {{0x11150, 0x1117F}},
5486 ['Mlym'] = {{0x0D00, 0x0D7F}},
5487 ['Mymr'] = {{0x1000, 0x109F}, {0xAA60, 0xAA7F}, {0xA9E0, 0xA9FF}},
5488 ['Orya'] = {{0x0B00, 0x0B7F}},
5489 ['Sinh'] = {{0x0D80, 0x0DFF}, {0x111E0, 0x111FF}},
5490 ['Syr'] = {{0x0700, 0x074F}, {0x0860, 0x086F}},
5491 ['Taml'] = {{0x0B80, 0x0BFF}},
5492 ['Telu'] = {{0x0C00, 0x0C7F}},
5493 ['Tfng'] = {{0x2D30, 0x2D7F}},
5494 ['Thai'] = {{0x0E00, 0x0E7F}},
5495 ['Tibt'] = {{0x0F00, 0x0FFF}},
5496 ['Vaii'] = {{0xA500, 0xA63F}},
5497 ['Yiii'] = {{0xA000, 0xA48F}, {0xA490, 0xA4CF}}
5498 }
5499
5500 Babel.script_blocks.Cyrs = Babel.script_blocks.Cyrl
5501 Babel.script_blocks.Hant = Babel.script_blocks.Hans
5502 Babel.script_blocks.Kana = Babel.script_blocks.Jpan
5503
5504 function Babel.locale_map(head)
5505   if not Babel.locale_mapped then return head end
5506
5507   local LOCALE = Babel.attr_locale
5508   local GLYPH = node.id('glyph')
5509   local inmath = false
5510   local toloc_save
5511   for item in node.traverse(head) do
5512     local toloc
5513     if not inmath and item.id == GLYPH then
5514       % Optimization: build a table with the chars found
5515       if Babel.chr_to_loc[item.char] then
5516         toloc = Babel.chr_to_loc[item.char]
5517       else
5518         for lc, maps in pairs(Babel.loc_to_scr) do
5519           for _, rg in pairs(maps) do
5520             if item.char >= rg[1] and item.char <= rg[2] then

```

```

5521         Babel.chr_to_loc[item.char] = lc
5522         toloc = lc
5523         break
5524     end
5525 end
5526 end
5527 end
5528 % Now, take action, but treat composite chars in a different
5529 % fashion, because they 'inherit' the previous locale. Not yet
5530 % optimized.
5531 if not toloc and
5532     (item.char >= 0x0300 and item.char <= 0x036F) or
5533     (item.char >= 0x1AB0 and item.char <= 0x1AFF) or
5534     (item.char >= 0x1DC0 and item.char <= 0x1DFF) then
5535     toloc = toloc_save
5536 end
5537 if toloc and toloc > -1 then
5538     if Babel.locale_props[toloc].lg then
5539         item.lang = Babel.locale_props[toloc].lg
5540         node.set_attribute(item, LOCALE, toloc)
5541     end
5542     if Babel.locale_props[toloc]['/'..item.font] then
5543         item.font = Babel.locale_props[toloc]['/'..item.font]
5544     end
5545     toloc_save = toloc
5546 end
5547 elseif not inmath and item.id == 7 then
5548     item.replace = item.replace and Babel.locale_map(item.replace)
5549     item.pre      = item.pre and Babel.locale_map(item.pre)
5550     item.post     = item.post and Babel.locale_map(item.post)
5551 elseif item.id == node.id'math' then
5552     inmath = (item.subtype == 0)
5553 end
5554 end
5555 return head
5556 end
5557 }

```

The code for \babelcharproperty is straightforward. Just note the modified lua table can be different.

```

5558 \newcommand\babelcharproperty[1]{%
5559   \count@=#1\relax
5560   \ifvmode
5561     \expandafter\bbl@chprop
5562   \else
5563     \bbl@error{\string\babelcharproperty\space can be used only in\\%
5564               vertical mode (preamble or between paragraphs)}%
5565     {See the manual for futher info}%
5566   \fi}
5567 \newcommand\bbl@chprop[3][\the\count@]{%
5568   \@tempcnta=#1\relax
5569   \bbl@ifunset{\bbl@chprop@#2}%
5570   {\bbl@error{No property named '#2'. Allowed values are\\%
5571             direction (bc), mirror (bmg), and linebreak (lb)}%
5572    {See the manual for futher info}}%
5573   }%
5574   \loop
5575     \bbl@cs{chprop@#2}{#3}%
5576     \ifnum\count@<\@tempcnta

```



```

5577 \advance\count@\ne
5578 \repeat}
5579 \def\bbl@chprop@direction#1{%
5580 \directlua{
5581   Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5582   Babel.characters[\the\count@]['d'] = '#1'
5583 }}
5584 \let\bbl@chprop@bc\bbl@chprop@direction
5585 \def\bbl@chprop@mirror#1{%
5586 \directlua{
5587   Babel.characters[\the\count@] = Babel.characters[\the\count@] or {}
5588   Babel.characters[\the\count@]['m'] = '\number#1'
5589 }}
5590 \let\bbl@chprop@bmg\bbl@chprop@mirror
5591 \def\bbl@chprop@linebreak#1{%
5592 \directlua{
5593   Babel.cjk_characters[\the\count@] = Babel.cjk_characters[\the\count@] or {}
5594   Babel.cjk_characters[\the\count@]['c'] = '#1'
5595 }}
5596 \let\bbl@chprop@lb\bbl@chprop@linebreak
5597 \def\bbl@chprop@locale#1{%
5598 \directlua{
5599   Babel.chr_to_loc = Babel.chr_to_loc or {}
5600   Babel.chr_to_loc[\the\count@] =
5601     \bbl@ifblank{#1}{-1000}{\the\bbl@cs{id@#1}}\space
5602 }}

```

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). The Lua code is below.

```

5603 \directlua{
5604   Babel.nohyphenation = \the\l@nohyphenation
5605 }

```

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  $\text{function}(m) \text{ return } m[1]..m[1]..'-' \text{ end}$ , where  $m$  are the matches returned after applying the pattern. With a mapped capture the functions are similar to  $\text{function}(m) \text{ return } \text{Babel.capt\_map}(m[1],1) \text{ end}$ , 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  $\backslash\text{directlua}$  does not take into account the current catcode of  $@$ , we just avoid this character in macro names (which explains the internal group, too).

```

5606 \begingroup
5607 \catcode`\~ = 12
5608 \catcode`\% = 12
5609 \catcode`\& = 14
5610 \gdef\babelposthyphenation#1#2#3{&%
5611   \bbl@activateposthyphen
5612   \begingroup
5613     \def\babeltempa{\bbl@add@list\babeltempb}&%
5614     \let\babeltempb\@empty
5615     \def\bbl@tempa{#3}&% TODO. Ugly trick to preserve {}:
5616     \bbl@replace\bbl@tempa{,}{ ,}&%
5617     \expandafter\bbl@foreach\expandafter{\bbl@tempa}{&%
5618       \bbl@ifsamestring{##1}{remove}&%
5619       {\bbl@add@list\babeltempb{nil}}&%
5620       {\directlua{
5621         local rep = {[##1]=]
5622         rep = rep:gsub('^%s*(remove)%s*$', 'remove = true')}

```

```

5623         rep = rep:gsub('^%s*(insert)%s*', ' ', 'insert = true, ')
5624         rep = rep:gsub(' (no)%s*=%s*([^\s,]*)', Babel.capture_func)
5625         rep = rep:gsub(' (pre)%s*=%s*([^\s,]*)', Babel.capture_func)
5626         rep = rep:gsub(' (post)%s*=%s*([^\s,]*)', Babel.capture_func)
5627         rep = rep:gsub('(string)%s*=%s*([^\s,]*)', Babel.capture_func)
5628         tex.print([[\\string\\babeltempa{[]] .. rep .. [[]]])
5629     }&&%
5630 \\directlua{
5631     local lbkr = Babel.linebreaking.replacements[1]
5632     local u = unicode.utf8
5633     local id = \\the\\csname l@#1\\endcsname
5634     &% Convert pattern:
5635     local patt = string.gsub([==[#2]==], '%s', ' ')
5636     if not u.find(patt, '()', nil, true) then
5637         patt = '()' .. patt .. '()'
5638     end
5639     patt = string.gsub(patt, '%(%)%\\', '^()')
5640     patt = string.gsub(patt, '%$(%)%', '()$')
5641     patt = u.gsub(patt, '{(.)}',
5642         function (n)
5643             return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
5644         end)
5645     patt = u.gsub(patt, '{(%x%x%x%x+)}',
5646         function (n)
5647             return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%1')
5648         end)
5649     lbkr[id] = lbkr[id] or {}
5650     table.insert(lbkr[id], { pattern = patt, replace = { \\babeltempb } })
5651 }&&%
5652 \\endgroup}
5653 % TODO. Copypaste pattern.
5654 \\gdef\\babelprehyphenation#1#2#3{&&%
5655 \\bbl@activateprehyphen
5656 \\begin{group}
5657 \\def\\babeltempa{\\bbl@add@list\\babeltempb}&&%
5658 \\let\\babeltempb\\@empty
5659 \\def\\bbl@tempa{#3}&&% TODO. Ugly trick to preserve {}:
5660 \\bbl@replace\\bbl@tempa{,}{ ,}&&%
5661 \\expandafter\\bbl@foreach\\expandafter{\\bbl@tempa}{&&%
5662     \\bbl@ifsamestring{##1}{remove}&&%
5663     {\\bbl@add@list\\babeltempb{nil}}&&%
5664     {\\directlua{
5665         local rep = [=[#1]=]
5666         rep = rep:gsub('^%s*(remove)%s*$ ', 'remove = true')
5667         rep = rep:gsub('^%s*(insert)%s*', ' ', 'insert = true, ')
5668         rep = rep:gsub('(string)%s*=%s*([^\s,]*)', Babel.capture_func)
5669         rep = rep:gsub('(space)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
5670             'space = { ' .. '%2, %3, %4' .. ' }')
5671         rep = rep:gsub('(spacefactor)%s*=%s*([%d%.]+)%s+([%d%.]+)%s+([%d%.]+)',
5672             'spacefactor = { ' .. '%2, %3, %4' .. ' }')
5673         rep = rep:gsub('(kashida)%s*=%s*([^\s,]*)', Babel.capture_kashida)
5674         tex.print([[\\string\\babeltempa{[]] .. rep .. [[]]])
5675     }&&%
5676 \\directlua{
5677     local lbkr = Babel.linebreaking.replacements[0]
5678     local u = unicode.utf8
5679     local id = \\the\\csname bbl@id@#1\\endcsname
5680     &% Convert pattern:
5681     local patt = string.gsub([==[#2]==], '%s', ' ')

```

```

5682     local patt = string.gsub(patt, '|', ' ')
5683     if not u.find(patt, '()', nil, true) then
5684         patt = '()' .. patt .. '()'
5685     end
5686     &% patt = string.gsub(patt, '%(%)%^', '^()')
5687     &% patt = string.gsub(patt, '([%^%])%$%(%)', '%1()$')
5688     patt = u.gsub(patt, '{(.)}',
5689         function (n)
5690             return '%' .. (tonumber(n) and (tonumber(n)+1) or n)
5691         end)
5692     patt = u.gsub(patt, '{(%x%x%x%x+)}',
5693         function (n)
5694             return u.gsub(u.char(tonumber(n, 16)), '(%p)', '%%%1')
5695         end)
5696     lbr[id] = lbr[id] or {}
5697     table.insert(lbr[id], { pattern = patt, replace = { \babeltempb } })
5698 }&%
5699 \endgroup}
5700 \endgroup
5701 \def\bbl@activateposthyphen{%
5702   \let\bbl@activateposthyphen\relax
5703   \directlua{
5704     require('babel-transforms.lua')
5705     Babel.linebreaking.add_after(Babel.post_hyphenate_replace)
5706   }}
5707 \def\bbl@activateprehyphen{%
5708   \let\bbl@activateprehyphen\relax
5709   \directlua{
5710     require('babel-transforms.lua')
5711     Babel.linebreaking.add_before(Babel.pre_hyphenate_replace)
5712   }}

```

### 13.9 Bidi

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  $\text{\LaTeX}$ . Just in case, consider the possibility it has not been loaded.

```

5713 \def\bbl@activate@preotf{%
5714   \let\bbl@activate@preotf\relax % only once
5715   \directlua{
5716     Babel = Babel or {}
5717     %
5718     function Babel.pre_otfload_v(head)
5719       if Babel.numbers and Babel.digits_mapped then
5720         head = Babel.numbers(head)
5721       end
5722       if Babel.bidi_enabled then
5723         head = Babel.bidi(head, false, dir)
5724       end
5725       return head
5726     end
5727     %
5728     function Babel.pre_otfload_h(head, gc, sz, pt, dir)
5729       if Babel.numbers and Babel.digits_mapped then
5730         head = Babel.numbers(head)
5731       end
5732       if Babel.bidi_enabled then
5733         head = Babel.bidi(head, false, dir)

```

```

5734     end
5735     return head
5736 end
5737 %
5738 luatexbase.add_to_callback('pre_linebreak_filter',
5739     Babel.pre_otfload_v,
5740     'Babel.pre_otfload_v',
5741     luatexbase.priority_in_callback('pre_linebreak_filter',
5742     'luaotfload.node_processor') or nil)
5743 %
5744 luatexbase.add_to_callback('hpack_filter',
5745     Babel.pre_otfload_h,
5746     'Babel.pre_otfload_h',
5747     luatexbase.priority_in_callback('hpack_filter',
5748     'luaotfload.node_processor') or nil)
5749 }}

```

The basic setup. The output is modified at a very low level to set the `\bodydir` to the `\pagedir`. Sadly, we have to deal with boxes in math with basic, so the `\bbl@mathboxdir` hack is activated every math with the package option `bidi`.

```

5750 \ifnum\bbl@bidimode>100 \ifnum\bbl@bidimode<200
5751 \let\bbl@beforeforeign\leavevmode
5752 \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
5753 \RequirePackage{luatexbase}
5754 \bbl@activate@preotf
5755 \directlua{
5756     require('babel-data-bidi.lua')
5757     \ifcase\expandafter\@gobbletwo\the\bbl@bidimode\or
5758         require('babel-bidi-basic.lua')
5759     \or
5760         require('babel-bidi-basic-r.lua')
5761     \fi}
5762 % TODO - to locale_props, not as separate attribute
5763 \newattribute\bbl@attr@dir
5764 \directlua{ Babel.attr_dir = luatexbase.registernumber'bbl@attr@dir' }
5765 % TODO. I don't like it, hackish:
5766 \bbl@exp{\output{\bodydir\pagedir\the\output}}
5767 \AtEndOfPackage{\EnableBabelHook{babel-bidi}}
5768 \fi\fi
5769 \chardef\bbl@thetextdir\z@
5770 \chardef\bbl@thepardir\z@
5771 \def\bbl@getluadir#1{%
5772     \directlua{
5773         if tex.#1dir == 'TLT' then
5774             tex.sprint('0')
5775         elseif tex.#1dir == 'TRT' then
5776             tex.sprint('1')
5777         end}}
5778 \def\bbl@setluadir#1#2#3{% 1=text/par.. 2=\textdir.. 3=0 lr/1 rl
5779     \ifcase#3\relax
5780         \ifcase\bbl@getluadir{#1}\relax\else
5781             #2 TLT\relax
5782         \fi
5783     \else
5784         \ifcase\bbl@getluadir{#1}\relax
5785             #2 TRT\relax
5786         \fi
5787     \fi}
5788 \def\bbl@textdir#1{%

```

```

5789 \bbl@setluadir{text}\texkdir{#1}%
5790 \chardef\bbl@thetexkdir#1\relax
5791 \setattribute\bbl@attr@dir{\numexpr\bbl@thepardir*3+#1}}
5792 \def\bbl@pardir#1{%
5793 \bbl@setluadir{par}\pardir{#1}%
5794 \chardef\bbl@thepardir#1\relax}
5795 \def\bbl@bodydir{\bbl@setluadir{body}\bodydir}
5796 \def\bbl@pagedir{\bbl@setluadir{page}\pagedir}
5797 \def\bbl@dirparastext{\pardir\the\texkdir\relax}% %%%
5798 %
5799 \ifnum\bbl@bidimode>\z@
5800 \def\bbl@mathboxdir{%
5801 \ifcase\bbl@thetexkdir\relax
5802 \everyhbox{\bbl@mathboxdir@aux L}%
5803 \else
5804 \everyhbox{\bbl@mathboxdir@aux R}%
5805 \fi}
5806 \def\bbl@mathboxdir@aux#1{%
5807 \@ifnextchar\egroup{}\{\texkdir T#1T\relax}}
5808 \frozen@everymath\expandafter{%
5809 \expandafter\bbl@mathboxdir\the\frozen@everymath}
5810 \frozen@everydisplay\expandafter{%
5811 \expandafter\bbl@mathboxdir\the\frozen@everydisplay}
5812 \fi

```

## 13.10 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`, `hline`, `colortbl`, `longtable`, `booktabs`, etc. However, `dcolumn` still fails.

```

5813 \bbl@trace{Redefinitions for bidi layout}
5814 \ifx\@eqnnum\undefined\else
5815 \ifx\bbl@attr@dir\undefined\else
5816 \edef\@eqnnum{%
5817 \unexpanded{\ifcase\bbl@attr@dir\else\bbl@texkdir\@ne\fi}%
5818 \unexpanded\expandafter{\@eqnnum}}
5819 \fi
5820 \fi
5821 \ifx\bbl@opt@layout\@nnil\endinput\fi % if no layout
5822 \ifnum\bbl@bidimode>\z@
5823 \def\bbl@nextfake#1{% non-local changes, use always inside a group!
5824 \bbl@exp{%
5825 \mathdir\the\bodydir
5826 #1% Once entered in math, set boxes to restore values
5827 \<ifmmode>%
5828 \everyvbox{%
5829 \the\everyvbox
5830 \bodydir\the\bodydir
5831 \mathdir\the\mathdir

```

```

5832     \everyhbox{\the\everyhbox}%
5833     \everyvbox{\the\everyvbox}}%
5834     \everyhbox{%
5835         \the\everyhbox
5836         \bodydir\the\bodydir
5837         \mathdir\the\mathdir
5838         \everyhbox{\the\everyhbox}%
5839         \everyvbox{\the\everyvbox}}%
5840     \<fi>}}%
5841 \def\@hangfrom#1{%
5842     \setbox\@tempboxa\hbox{{#1}}%
5843     \hangindent\wd\@tempboxa
5844     \ifnum\bb1@getluadir{page}=\bb1@getluadir{par}\else
5845         \shapemode\@ne
5846     \fi
5847     \noindent\box\@tempboxa}
5848 \fi
5849 \IfBabelLayout{tabular}
5850 {\let\bb1@OL@@tabular\@tabular
5851  \bb1@replace\@tabular{$}{\bb1@nextfake$}%
5852  \let\bb1@NL@@tabular\@tabular
5853  \AtBeginDocument{%
5854      \ifx\bb1@NL@@tabular\@tabular\else
5855          \bb1@replace\@tabular{$}{\bb1@nextfake$}%
5856          \let\bb1@NL@@tabular\@tabular
5857      \fi}}
5858 {}
5859 \IfBabelLayout{lists}
5860 {\let\bb1@OL@list\list
5861  \bb1@sreplace\list{\parshape}{\bb1@listparshape}%
5862  \let\bb1@NL@list\list
5863  \def\bb1@listparshape#1#2#3{%
5864      \parshape #1 #2 #3 %
5865      \ifnum\bb1@getluadir{page}=\bb1@getluadir{par}\else
5866          \shapemode\tw@
5867      \fi}}
5868 {}
5869 \IfBabelLayout{graphics}
5870 {\let\bb1@pictresetdir\relax
5871  \def\bb1@pictsetdir#1{%
5872      \ifcase\bb1@thetextdir
5873          \let\bb1@pictresetdir\relax
5874      \else
5875          \ifcase#1\bodydir TLT % Remember this sets the inner boxes
5876              \or\textdir TLT
5877              \else\bodydir TLT \textdir TLT
5878          \fi
5879          % \text\par\dir required in pgf:
5880          \def\bb1@pictresetdir{\bodydir TRT\pardir TRT\textdir TRT\relax}%
5881      \fi}%
5882  \ifx\AddToHook\undefined\else
5883      \AddToHook{env/picture/begin}{\bb1@pictsetdir\tw@}%
5884      \directlua{
5885          Babel.get_picture_dir = true
5886          Babel.picture_has_bidi = 0
5887          function Babel.picture_dir (head)
5888              if not Babel.get_picture_dir then return head end
5889              for item in node.traverse(head) do
5890                  if item.id == node.id'glyph' then

```

```

5891         local itemchar = item.char
5892         % TODO. Copypaste pattern from Babel.bidi (-r)
5893         local chardata = Babel.characters[itemchar]
5894         local dir = chardata and chardata.d or nil
5895         if not dir then
5896             for nn, et in ipairs(Babel.ranges) do
5897                 if itemchar < et[1] then
5898                     break
5899                 elseif itemchar <= et[2] then
5900                     dir = et[3]
5901                     break
5902                 end
5903             end
5904         end
5905         if dir and (dir == 'al' or dir == 'r') then
5906             Babel.picture_has_bidi = 1
5907         end
5908     end
5909 end
5910 return head
5911 end
5912 luatexbase.add_to_callback("hpack_filter", Babel.picture_dir,
5913     "Babel.picture_dir")
5914 }%
5915 \AtBeginDocument{%
5916     \long\def\put(#1,#2)#3{%
5917         \@killglue
5918         % Try:
5919         \ifx\bbbl@pictresetdir\relax
5920             \def\bbbl@tempc{0}%
5921         \else
5922             \directlua{
5923                 Babel.get_picture_dir = true
5924                 Babel.picture_has_bidi = 0
5925             }%
5926             \setbox\z@\hb@xt@\z@{%
5927                 \@defaultunitsset\@tempdimc{#1}\unitlength
5928                 \kern\@tempdimc
5929                 #3\hss}%
5930             \edef\bbbl@tempc{\directlua{tex.print(Babel.picture_has_bidi)}}%
5931         \fi
5932         % Do:
5933         \@defaultunitsset\@tempdimc{#2}\unitlength
5934         \raise\@tempdimc\hb@xt@\z@{%
5935             \@defaultunitsset\@tempdimc{#1}\unitlength
5936             \kern\@tempdimc
5937             {\ifnum\bbbl@tempc>\z@\bbbl@pictresetdir\fi#3}\hss}%
5938         \ignorespaces}%
5939         \MakeRobust\put}%
5940 \fi
5941 \AtBeginDocument
5942     {\ifx\tikz@atbegin@node\undefined\else
5943         \ifx\AddToHook\undefined\else % TODO. Still tentative.
5944             \AddToHook{env/pgfpicture/begin}{\bbbl@pictsetdir\@ne}%
5945             \bbbl@add\pgfinterruptpicture{\bbbl@pictresetdir}%
5946         \fi
5947         \let\bbbl@OL@pgfpicture\pgfpicture
5948         \bbbl@sreplace\pgfpicture{\pgfpicturetrue}%
5949         {\bbbl@pictsetdir\z@\pgfpicturetrue}%

```

```

5950      \bbl@add\pgfsys@beginpicture{\bbl@pictsetdir\z}%
5951      \bbl@add\tikz@atbegin@node{\bbl@pictresetdir}%
5952      \bbl@sreplace\tikz{\begingroup}%
5953      {\begingroup\bbl@pictsetdir\tw}%
5954      \fi
5955      \ifx\AddToHook\undefined\else
5956      \AddToHook{env/tcolorbox/begin}{\bbl@pictsetdir\ne}%
5957      \fi
5958      }}
5959  {}

```

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`.

```

5960 \IfBabelLayout{counters}%
5961  {\let\bbl@OL@@textsuperscript\textsuperscript
5962   \bbl@sreplace\textsuperscript{\m@th}{\m@th\mathdir\pagedir}%
5963   \let\bbl@latinarabic=\@arabic
5964   \let\bbl@OL@@arabic\@arabic
5965   \def\@arabic#1{\babelsublr{\bbl@latinarabic#1}}%
5966   \@ifpackagewith{babel}{bidi=default}%
5967   {\let\bbl@asciroman=\@roman
5968    \let\bbl@OL@@roman\@roman
5969    \def\@roman#1{\babelsublr{\ensureascii{\bbl@asciroman#1}}}%
5970    \let\bbl@asciiRoman=\@Roman
5971    \let\bbl@OL@@roman\@Roman
5972    \def\@Roman#1{\babelsublr{\ensureascii{\bbl@asciiRoman#1}}}%
5973    \let\bbl@OL@labelenumii\labelenumii
5974    \def\labelenumii{}\theenumii}%
5975    \let\bbl@OL@p@enumiii\p@enumiii
5976    \def\p@enumiii{\p@enumii}\theenumii{}\{\}\{\}}
5977  <<Footnote changes>>
5978 \IfBabelLayout{footnotes}%
5979  {\let\bbl@OL@footnote\footnote
5980   \BabelFootnote\footnote\language\{}}%
5981   \BabelFootnote\localfootnote\language\{}}%
5982   \BabelFootnote\mainfootnote\{}}%
5983  {}

```

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.

```

5984 \IfBabelLayout{extras}%
5985  {\let\bbl@OL@underline\underline
5986   \bbl@sreplace\underline{\$@@@underline}{\bbl@nextfake\$@@@underline}%
5987   \let\bbl@OL@LaTeX2e\LaTeX2e
5988   \DeclareRobustCommand{\LaTeXe}{\mbox{\m@th
5989    \if b\expandafter\@car\@series\@nil\boldmath\fi
5990    \babelsublr{%
5991     \LaTeX\kern.15em2\bbl@nextfake$_{\textstyle\varepsilon}$}}}
5992   {}
5993 </luatex>

```

### 13.11 Lua: transforms

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.

```

5994 (*transforms)
5995 Babel.linebreaking.replacements = {}
5996 Babel.linebreaking.replacements[0] = {} -- pre
5997 Babel.linebreaking.replacements[1] = {} -- post
5998
5999 -- Discretionaries contain strings as nodes
6000 function Babel.str_to_nodes(fn, matches, base)
6001   local n, head, last
6002   if fn == nil then return nil end
6003   for s in string.utfvalues(fn(matches)) do
6004     if base.id == 7 then
6005       base = base.replace
6006     end
6007     n = node.copy(base)
6008     n.char = s
6009     if not head then
6010       head = n
6011     else
6012       last.next = n
6013     end
6014     last = n
6015   end
6016   return head
6017 end
6018
6019 Babel.fetch_subtext = {}
6020
6021 Babel.ignore_pre_char = function(node)
6022   return (node.lang == Babel.nohyphenation)
6023 end
6024
6025 -- Merging both functions doesn't seem feasible, because there are too
6026 -- many differences.
6027 Babel.fetch_subtext[0] = function(head)
6028   local word_string = ''
6029   local word_nodes = {}
6030   local lang
6031   local item = head
6032   local inmath = false
6033
6034   while item do
6035     if item.id == 11 then
6036       inmath = (item.subtype == 0)
6037     end
6038
6039     if inmath then
6040       -- pass
6041     elseif item.id == 29 then
6042       local locale = node.get_attribute(item, Babel.attr_locale)
6043     end
6044   end
6045 end

```

```

6046     if lang == locale or lang == nil then
6047         lang = lang or locale
6048         if Babel.ignore_pre_char(item) then
6049             word_string = word_string .. Babel.us_char
6050         else
6051             word_string = word_string .. unicode.utf8.char(item.char)
6052         end
6053         word_nodes[#word_nodes+1] = item
6054     else
6055         break
6056     end
6057
6058 elseif item.id == 12 and item.subtype == 13 then
6059     word_string = word_string .. ' '
6060     word_nodes[#word_nodes+1] = item
6061
6062     -- Ignore leading unrecognized nodes, too.
6063 elseif word_string ~= '' then
6064     word_string = word_string .. Babel.us_char
6065     word_nodes[#word_nodes+1] = item -- Will be ignored
6066 end
6067
6068 item = item.next
6069 end
6070
6071 -- Here and above we remove some trailing chars but not the
6072 -- corresponding nodes. But they aren't accessed.
6073 if word_string:sub(-1) == ' ' then
6074     word_string = word_string:sub(1,-2)
6075 end
6076 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
6077 return word_string, word_nodes, item, lang
6078 end
6079
6080 Babel.fetch_subtext[1] = function(head)
6081     local word_string = ''
6082     local word_nodes = {}
6083     local lang
6084     local item = head
6085     local inmath = false
6086
6087     while item do
6088
6089         if item.id == 11 then
6090             inmath = (item.subtype == 0)
6091         end
6092
6093         if inmath then
6094             -- pass
6095
6096         elseif item.id == 29 then
6097             if item.lang == lang or lang == nil then
6098                 if (item.char ~= 124) and (item.char ~= 61) then -- not =, not |
6099                     lang = lang or item.lang
6100                     word_string = word_string .. unicode.utf8.char(item.char)
6101                     word_nodes[#word_nodes+1] = item
6102                 end
6103             else
6104                 break

```

```

6105     end
6106
6107     elseif item.id == 7 and item.subtype == 2 then
6108         word_string = word_string .. '='
6109         word_nodes[#word_nodes+1] = item
6110
6111     elseif item.id == 7 and item.subtype == 3 then
6112         word_string = word_string .. '|'
6113         word_nodes[#word_nodes+1] = item
6114
6115     -- (1) Go to next word if nothing was found, and (2) implicitly
6116     -- remove leading USs.
6117     elseif word_string == '' then
6118         -- pass
6119
6120     -- This is the responsible for splitting by words.
6121     elseif (item.id == 12 and item.subtype == 13) then
6122         break
6123
6124     else
6125         word_string = word_string .. Babel.us_char
6126         word_nodes[#word_nodes+1] = item -- Will be ignored
6127     end
6128
6129     item = item.next
6130 end
6131
6132 word_string = unicode.utf8.gsub(word_string, Babel.us_char .. '+$', '')
6133 return word_string, word_nodes, item, lang
6134 end
6135
6136 function Babel.pre_hyphenate_replace(head)
6137     Babel.hyphenate_replace(head, 0)
6138 end
6139
6140 function Babel.post_hyphenate_replace(head)
6141     Babel.hyphenate_replace(head, 1)
6142 end
6143
6144 Babel.us_char = string.char(31)
6145
6146 function Babel.hyphenate_replace(head, mode)
6147     local u = unicode.utf8
6148     local lbkr = Babel.linebreaking.replacements[mode]
6149
6150     local word_head = head
6151
6152     while true do -- for each subtext block
6153
6154         local w, w_nodes, nw, lang = Babel.fetch_subtext[mode](word_head)
6155
6156         if Babel.debug then
6157             print()
6158             print((mode == 0) and '@@@<' or '@@@>', w)
6159         end
6160
6161         if nw == nil and w == '' then break end
6162
6163         if not lang then goto next end

```

```

6164 if not lbkr[lang] then goto next end
6165
6166 -- For each saved (pre|post)hyphenation. TODO. Reconsider how
6167 -- loops are nested.
6168 for k=1, #lbkr[lang] do
6169     local p = lbkr[lang][k].pattern
6170     local r = lbkr[lang][k].replace
6171
6172     if Babel.debug then
6173         print('*****', p, mode)
6174     end
6175
6176     -- This variable is set in some cases below to the first *byte*
6177     -- after the match, either as found by u.match (faster) or the
6178     -- computed position based on sc if w has changed.
6179     local last_match = 0
6180     local step = 0
6181
6182     -- For every match.
6183     while true do
6184         if Babel.debug then
6185             print('====')
6186         end
6187         local new -- used when inserting and removing nodes
6188
6189         local matches = { u.match(w, p, last_match) }
6190
6191         if #matches < 2 then break end
6192
6193         -- Get and remove empty captures (with ()'s, which return a
6194         -- number with the position), and keep actual captures
6195         -- (from (...)), if any, in matches.
6196         local first = table.remove(matches, 1)
6197         local last = table.remove(matches, #matches)
6198         -- Non re-fetched substrings may contain \31, which separates
6199         -- subsubstrings.
6200         if string.find(w:sub(first, last-1), Babel.us_char) then break end
6201
6202         local save_last = last -- with A()BC()D, points to D
6203
6204         -- Fix offsets, from bytes to unicode. Explained above.
6205         first = u.len(w:sub(1, first-1)) + 1
6206         last = u.len(w:sub(1, last-1)) -- now last points to C
6207
6208         -- This loop stores in n small table the nodes
6209         -- corresponding to the pattern. Used by 'data' to provide a
6210         -- predictable behavior with 'insert' (now w_nodes is modified on
6211         -- the fly), and also access to 'remove'd nodes.
6212         local sc = first-1 -- Used below, too
6213         local data_nodes = {}
6214
6215         for q = 1, last-first+1 do
6216             data_nodes[q] = w_nodes[sc+q]
6217         end
6218
6219         -- This loop traverses the matched substring and takes the
6220         -- corresponding action stored in the replacement list.
6221         -- sc = the position in substr nodes / string
6222         -- rc = the replacement table index

```

```

6223     local rc = 0
6224
6225     while rc < last-first+1 do -- for each replacement
6226         if Babel.debug then
6227             print('.....', rc + 1)
6228         end
6229         sc = sc + 1
6230         rc = rc + 1
6231
6232         if Babel.debug then
6233             Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6234             local ss = ''
6235             for itt in node.traverse(head) do
6236                 if itt.id == 29 then
6237                     ss = ss .. unicode.utf8.char(itt.char)
6238                 else
6239                     ss = ss .. '{' .. itt.id .. '}'
6240                 end
6241             end
6242             print('*****', ss)
6243         end
6244
6245         local crep = r[rc]
6246         local item = w_nodes[sc]
6247         local item_base = item
6248         local placeholder = Babel.us_char
6249         local d
6250
6251         if crep and crep.data then
6252             item_base = data_nodes[crep.data]
6253         end
6254
6255         if crep then
6256             step = crep.step or 0
6257         end
6258
6259         if crep and next(crep) == nil then -- = {}
6260             last_match = save_last -- Optimization
6261             goto next
6262         end
6263
6264         elseif crep == nil or crep.remove then
6265             node.remove(head, item)
6266             table.remove(w_nodes, sc)
6267             w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
6268             sc = sc - 1 -- Nothing has been inserted.
6269             last_match = utf8.offset(w, sc+1+step)
6270             goto next
6271
6272         elseif crep and crep.kashida then -- Experimental
6273             node.set_attribute(item,
6274                 Babel.attr_kashida,
6275                 crep.kashida)
6276             last_match = utf8.offset(w, sc+1+step)
6277             goto next
6278
6279         elseif crep and crep.string then
6280             local str = crep.string(matches)
6281             if str == '' then -- Gather with nil

```

```

6282         node.remove(head, item)
6283         table.remove(w_nodes, sc)
6284         w = u.sub(w, 1, sc-1) .. u.sub(w, sc+1)
6285         sc = sc - 1 -- Nothing has been inserted.
6286     else
6287         local loop_first = true
6288         for s in string.utfvalues(str) do
6289             d = node.copy(item_base)
6290             d.char = s
6291             if loop_first then
6292                 loop_first = false
6293                 head, new = node.insert_before(head, item, d)
6294                 if sc == 1 then
6295                     word_head = head
6296                 end
6297                 w_nodes[sc] = d
6298                 w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc+1)
6299             else
6300                 sc = sc + 1
6301                 head, new = node.insert_before(head, item, d)
6302                 table.insert(w_nodes, sc, new)
6303                 w = u.sub(w, 1, sc-1) .. u.char(s) .. u.sub(w, sc)
6304             end
6305             if Babel.debug then
6306                 print('.....', 'str')
6307                 Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6308             end
6309         end -- for
6310         node.remove(head, item)
6311     end -- if ''
6312     last_match = utf8.offset(w, sc+1+step)
6313     goto next
6314
6315 elseif mode == 1 and crep and (crep.pre or crep.no or crep.post) then
6316     d = node.new(7, 0) -- (disc, discretionary)
6317     d.pre = Babel.str_to_nodes(crep.pre, matches, item_base)
6318     d.post = Babel.str_to_nodes(crep.post, matches, item_base)
6319     d.replace = Babel.str_to_nodes(crep.no, matches, item_base)
6320     d.attr = item_base.attr
6321     if crep.pre == nil then -- TeXbook p96
6322         d.penalty = crep.penalty or tex.hyphenpenalty
6323     else
6324         d.penalty = crep.penalty or tex.exhyphenpenalty
6325     end
6326     placeholder = '|'
6327     head, new = node.insert_before(head, item, d)
6328
6329 elseif mode == 0 and crep and (crep.pre or crep.no or crep.post) then
6330     -- ERROR
6331
6332 elseif crep and crep.penalty then
6333     d = node.new(14, 0) -- (penalty, userpenalty)
6334     d.attr = item_base.attr
6335     d.penalty = crep.penalty
6336     head, new = node.insert_before(head, item, d)
6337
6338 elseif crep and crep.space then
6339     -- 655360 = 10 pt = 10 * 65536 sp
6340     d = node.new(12, 13) -- (glue, spaceskip)

```

```

6341         local quad = font.getfont(item_base.font).size or 655360
6342         node.setglue(d, crep.space[1] * quad,
6343                     crep.space[2] * quad,
6344                     crep.space[3] * quad)
6345         if mode == 0 then
6346             placeholder = ' '
6347         end
6348         head, new = node.insert_before(head, item, d)
6349
6350     elseif crep and crep.spacefactor then
6351         d = node.new(12, 13) -- (glue, spaceskip)
6352         local base_font = font.getfont(item_base.font)
6353         node.setglue(d,
6354                     crep.spacefactor[1] * base_font.parameters['space'],
6355                     crep.spacefactor[2] * base_font.parameters['space_stretch'],
6356                     crep.spacefactor[3] * base_font.parameters['space_shrink'])
6357         if mode == 0 then
6358             placeholder = ' '
6359         end
6360         head, new = node.insert_before(head, item, d)
6361
6362     elseif mode == 0 and crep and crep.space then
6363         -- ERROR
6364
6365     end -- ie replacement cases
6366
6367     -- Shared by disc, space and penalty.
6368     if sc == 1 then
6369         word_head = head
6370     end
6371     if crep.insert then
6372         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc)
6373         table.insert(w_nodes, sc, new)
6374         last = last + 1
6375     else
6376         w_nodes[sc] = d
6377         node.remove(head, item)
6378         w = u.sub(w, 1, sc-1) .. placeholder .. u.sub(w, sc+1)
6379     end
6380
6381     last_match = utf8.offset(w, sc+1+step)
6382
6383     ::next::
6384
6385     end -- for each replacement
6386
6387     if Babel.debug then
6388         print('.....', '/')
6389         Babel.debug_hyph(w, w_nodes, sc, first, last, last_match)
6390     end
6391
6392     end -- for match
6393
6394     end -- for patterns
6395
6396     ::next::
6397     word_head = nw
6398     end -- for substring
6399     return head

```

```

6400 end
6401
6402 -- This table stores capture maps, numbered consecutively
6403 Babel.capture_maps = {}
6404
6405 -- The following functions belong to the next macro
6406 function Babel.capture_func(key, cap)
6407   local ret = "[" .. cap:gsub('{{[0-9]}}', "]]..m[%1]..[" .. "]"
6408   local cnt
6409   local u = unicode.utf8
6410   ret, cnt = ret:gsub('{{[0-9]}|([^\]]+)|(.-)}', Babel.capture_func_map)
6411   if cnt == 0 then
6412     ret = u.gsub(ret, '{{(%x%x%x%x+)}}',
6413       function (n)
6414         return u.char(tonumber(n, 16))
6415       end)
6416   end
6417   ret = ret:gsub("%[%[%]]%.", '')
6418   ret = ret:gsub("%.%[%[%]]%", '')
6419   return key .. [[=function(m) return ]] .. ret .. [[ end]]
6420 end
6421
6422 function Babel.capt_map(from, mapno)
6423   return Babel.capture_maps[mapno][from] or from
6424 end
6425
6426 -- Handle the {n|abc|ABC} syntax in captures
6427 function Babel.capture_func_map(capno, from, to)
6428   local u = unicode.utf8
6429   from = u.gsub(from, '{{(%x%x%x%x+)}}',
6430     function (n)
6431       return u.char(tonumber(n, 16))
6432     end)
6433   to = u.gsub(to, '{{(%x%x%x%x+)}}',
6434     function (n)
6435       return u.char(tonumber(n, 16))
6436     end)
6437   local froms = {}
6438   for s in string.utfcharacters(from) do
6439     table.insert(froms, s)
6440   end
6441   local cnt = 1
6442   table.insert(Babel.capture_maps, {})
6443   local mlen = table.getn(Babel.capture_maps)
6444   for s in string.utfcharacters(to) do
6445     Babel.capture_maps[mlen][froms[cnt]] = s
6446     cnt = cnt + 1
6447   end
6448   return "]]..Babel.capt_map(m[" .. capno .. "], " ..
6449     (mlen) .. ").." .. "["
6450 end
6451
6452 -- Create/Extend reversed sorted list of kashida weights:
6453 function Babel.capture_kashida(key, wt)
6454   wt = tonumber(wt)
6455   if Babel.kashida_wts then
6456     for p, q in ipairs(Babel.kashida_wts) do
6457       if wt == q then
6458         break

```



```

6459     elseif wt > q then
6460         table.insert(Babel.kashida_wts, p, wt)
6461         break
6462     elseif table.getn(Babel.kashida_wts) == p then
6463         table.insert(Babel.kashida_wts, wt)
6464     end
6465 end
6466 else
6467     Babel.kashida_wts = { wt }
6468 end
6469 return 'kashida = ' .. wt
6470 end
6471 </transforms>

```

### 13.12 Lua: Auto bidi with basic and basic-r

The file `babel-data-bidi.lua` currently only contains data. It is a large and boring file and it is not shown here (see the generated file), but here is a sample:

```

[0x25]={d='et'},
[0x26]={d='on'},
[0x27]={d='on'},
[0x28]={d='on', m=0x29},
[0x29]={d='on', m=0x28},
[0x2A]={d='on'},
[0x2B]={d='es'},
[0x2C]={d='cs'},

```

For the meaning of these codes, see the Unicode standard.

Now the `basic-r` bidi mode. One of the aims is to implement a fast and simple bidi algorithm, with a single loop. I managed to do it for R texts, with a second smaller loop for a special case. The code is still somewhat chaotic, but its behavior is essentially correct. I cannot resist copying the following text from Emacs `bidi.c` (which also attempts to implement the bidi algorithm with a single loop):

Arrrgh!! The UAX#9 algorithm is too deeply entrenched in the assumption of batch-style processing [...]. May the fleas of a thousand camels infest the armpits of those who design supposedly general-purpose algorithms by looking at their own implementations, and fail to consider other possible implementations!

Well, it took me some time to guess what the batch rules in UAX#9 actually mean (in other word, *what* they do and *why*, and not only *how*), but I think (or I hope) I've managed to understand them. In some sense, there are two bidi modes, one for numbers, and the other for text. Furthermore, setting just the direction in R text is not enough, because there are actually *two* R modes (set explicitly in Unicode with RLM and ALM). In `babel` the `dir` is set by a higher protocol based on the language/script, which in turn sets the correct `dir` (<l>, <r> or <al>).

From UAX#9: “Where available, markup should be used instead of the explicit formatting characters”. So, this simple version just ignores formatting characters. Actually, most of that annex is devoted to how to handle them.

BD14-BD16 are not implemented. Unicode (and the W3C) are making a great effort to deal with some special problematic cases in “streamed” plain text. I don’t think this is the way to go – particular issues should be fixed by a high level interface taking into account the needs of the document. And here is where `luatex` excels, because everything related to bidi writing is under our control.

```

6472 (*basic-r)
6473 Babel = Babel or {}
6474
6475 Babel.bidi_enabled = true
6476
6477 require('babel-data-bidi.lua')

```

```

6478
6479 local characters = Babel.characters
6480 local ranges = Babel.ranges
6481
6482 local DIR = node.id("dir")
6483
6484 local function dir_mark(head, from, to, outer)
6485   dir = (outer == 'r') and 'TLT' or 'TRT' -- ie, reverse
6486   local d = node.new(DIR)
6487   d.dir = '+' .. dir
6488   node.insert_before(head, from, d)
6489   d = node.new(DIR)
6490   d.dir = '-' .. dir
6491   node.insert_after(head, to, d)
6492 end
6493
6494 function Babel.bidi(head, ispar)
6495   local first_n, last_n          -- first and last char with nums
6496   local last_es                  -- an auxiliary 'last' used with nums
6497   local first_d, last_d          -- first and last char in L/R block
6498   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):

```

6499   local strong = ('TRT' == tex.pardir) and 'r' or 'l'
6500   local strong_lr = (strong == 'l') and 'l' or 'r'
6501   local outer = strong
6502
6503   local new_dir = false
6504   local first_dir = false
6505   local inmath = false
6506
6507   local last_lr
6508
6509   local type_n = ''
6510
6511   for item in node.traverse(head) do
6512     -- three cases: glyph, dir, otherwise
6513     if item.id == node.id('glyph')
6514       or (item.id == 7 and item.subtype == 2) then
6515       local itemchar
6516       if item.id == 7 and item.subtype == 2 then
6517         itemchar = item.replace.char
6518       else
6519         itemchar = item.char
6520       end
6521       local chardata = characters[itemchar]
6522       dir = chardata and chardata.d or nil
6523       if not dir then
6524         for nn, et in ipairs(ranges) do
6525           if itemchar < et[1] then
6526             break
6527           elseif itemchar <= et[2] then
6528             dir = et[3]
6529             break
6530           end
6531         end
6532       end

```

```

6533     end
6534     end
6535     dir = dir or 'l'
6536     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).

```

6537     if new_dir then
6538         attr_dir = 0
6539         for at in node.traverse(item.attr) do
6540             if at.number == Babel.attr_dir then
6541                 attr_dir = at.value % 3
6542             end
6543         end
6544         if attr_dir == 1 then
6545             strong = 'r'
6546         elseif attr_dir == 2 then
6547             strong = 'al'
6548         else
6549             strong = 'l'
6550         end
6551         strong_lr = (strong == 'l') and 'l' or 'r'
6552         outer = strong_lr
6553         new_dir = false
6554     end
6555
6556     if dir == 'nsm' then dir = strong end -- W1

```

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

```

6557     dir_real = dir -- We need dir_real to set strong below
6558     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:

```

6559     if strong == 'al' then
6560         if dir == 'en' then dir = 'an' end -- W2
6561         if dir == 'et' or dir == 'es' then dir = 'on' end -- W6
6562         strong_lr = 'r' -- W3
6563     end

```

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

```

6564     elseif item.id == node.id'dir' and not inmath then
6565         new_dir = true
6566         dir = nil
6567     elseif item.id == node.id'math' then
6568         inmath = (item.subtype == 0)
6569     else
6570         dir = nil -- Not a char
6571     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>.

```

6572     if dir == 'en' or dir == 'an' or dir == 'et' then
6573         if dir ~= 'et' then

```

```

6574         type_n = dir
6575     end
6576     first_n = first_n or item
6577     last_n = last_es or item
6578     last_es = nil
6579     elseif dir == 'es' and last_n then -- W3+W6
6580         last_es = item
6581     elseif dir == 'cs' then -- it's right - do nothing
6582     elseif first_n then -- & if dir = any but en, et, an, es, cs, inc nil
6583         if strong_lr == 'r' and type_n ~= '' then
6584             dir_mark(head, first_n, last_n, 'r')
6585         elseif strong_lr == 'l' and first_d and type_n == 'an' then
6586             dir_mark(head, first_n, last_n, 'r')
6587             dir_mark(head, first_d, last_d, outer)
6588             first_d, last_d = nil, nil
6589         elseif strong_lr == 'l' and type_n ~= '' then
6590             last_d = last_n
6591         end
6592         type_n = ''
6593         first_n, last_n = nil, nil
6594     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:

```

6595     if dir == 'l' or dir == 'r' then
6596         if dir ~= outer then
6597             first_d = first_d or item
6598             last_d = item
6599         elseif first_d and dir ~= strong_lr then
6600             dir_mark(head, first_d, last_d, outer)
6601             first_d, last_d = nil, nil
6602         end
6603     end

```

**Mirroring.** Each chunk of text in a certain language is considered a “closed” sequence. If <r on r> and <l on l>, it's clearly <r> and <l>, resptly, but with other combinations depends on outer. From all these, we select only those resolving <on> → <r>. At the beginning (when last\_lr is nil) of an R text, they are mirrored directly.

TODO - numbers in R mode are processed. It doesn't hurt, but should not be done.

```

6604     if dir and not last_lr and dir ~= 'l' and outer == 'r' then
6605         item.char = characters[item.char] and
6606             characters[item.char].m or item.char
6607     elseif (dir or new_dir) and last_lr ~= item then
6608         local mir = outer .. strong_lr .. (dir or outer)
6609         if mir == 'rrr' or mir == 'lrr' or mir == 'rrl' or mir == 'rlr' then
6610             for ch in node.traverse(node.next(last_lr)) do
6611                 if ch == item then break end
6612                 if ch.id == node.id'glyph' and characters[ch.char] then
6613                     ch.char = characters[ch.char].m or ch.char
6614                 end
6615             end
6616         end
6617     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).

```

6618     if dir == 'l' or dir == 'r' then

```

```

6619     last_lr = item
6620     strong = dir_real           -- Don't search back - best save now
6621     strong_lr = (strong == 'l') and 'l' or 'r'
6622     elseif new_dir then
6623         last_lr = nil
6624     end
6625 end

```

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

```

6626 if last_lr and outer == 'r' then
6627     for ch in node.traverse_id(node.id'glyph', node.next(last_lr)) do
6628         if characters[ch.char] then
6629             ch.char = characters[ch.char].m or ch.char
6630         end
6631     end
6632 end
6633 if first_n then
6634     dir_mark(head, first_n, last_n, outer)
6635 end
6636 if first_d then
6637     dir_mark(head, first_d, last_d, outer)
6638 end

```

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

```

6639 return node.prev(head) or head
6640 end
6641 </basic-r>

```

And here the Lua code for bidi=basic:

```

6642 <*basic>
6643 Babel = Babel or {}
6644
6645 -- eg, Babel.fontmap[1][<prefontid>]=<dirfontid>
6646
6647 Babel.fontmap = Babel.fontmap or {}
6648 Babel.fontmap[0] = {}      -- l
6649 Babel.fontmap[1] = {}      -- r
6650 Babel.fontmap[2] = {}      -- al/an
6651
6652 Babel.bidi_enabled = true
6653 Babel.mirroring_enabled = true
6654
6655 require('babel-data-bidi.lua')
6656
6657 local characters = Babel.characters
6658 local ranges = Babel.ranges
6659
6660 local DIR = node.id('dir')
6661 local GLYPH = node.id('glyph')
6662
6663 local function insert_implicit(head, state, outer)
6664     local new_state = state
6665     if state.sim and state.eim and state.sim ~= state.eim then
6666         dir = ((outer == 'r') and 'TLT' or 'TRT') -- ie, reverse
6667         local d = node.new(DIR)
6668         d.dir = '+' .. dir
6669         node.insert_before(head, state.sim, d)
6670         local d = node.new(DIR)

```

```

6671     d.dir = '-' .. dir
6672     node.insert_after(head, state.eim, d)
6673 end
6674 new_state.sim, new_state.eim = nil, nil
6675 return head, new_state
6676 end
6677
6678 local function insert_numeric(head, state)
6679     local new
6680     local new_state = state
6681     if state.san and state.ean and state.san ~= state.ean then
6682         local d = node.new(DIR)
6683         d.dir = '+TLT'
6684         _, new = node.insert_before(head, state.san, d)
6685         if state.san == state.sim then state.sim = new end
6686         local d = node.new(DIR)
6687         d.dir = '-TLT'
6688         _, new = node.insert_after(head, state.ean, d)
6689         if state.ean == state.eim then state.eim = new end
6690     end
6691     new_state.san, new_state.ean = nil, nil
6692     return head, new_state
6693 end
6694
6695 -- TODO - \hbox with an explicit dir can lead to wrong results
6696 -- <R \hbox dir TLT{<R>}> and <L \hbox dir TRT{<L>}>. A small attempt
6697 -- was s made to improve the situation, but the problem is the 3-dir
6698 -- model in babel/Unicode and the 2-dir model in LuaTeX don't fit
6699 -- well.
6700
6701 function Babel.bidi(head, ispar, hdir)
6702     local d -- d is used mainly for computations in a loop
6703     local prev_d = ''
6704     local new_d = false
6705
6706     local nodes = {}
6707     local outer_first = nil
6708     local inmath = false
6709
6710     local glue_d = nil
6711     local glue_i = nil
6712
6713     local has_en = false
6714     local first_et = nil
6715
6716     local ATDIR = Babel.attr_dir
6717
6718     local save_outer
6719     local temp = node.get_attribute(head, ATDIR)
6720     if temp then
6721         temp = temp % 3
6722         save_outer = (temp == 0 and 'l') or
6723                     (temp == 1 and 'r') or
6724                     (temp == 2 and 'al')
6725     elseif ispar then -- Or error? Shouldn't happen
6726         save_outer = ('TRT' == tex.pardir) and 'r' or 'l'
6727     else -- Or error? Shouldn't happen
6728         save_outer = ('TRT' == hdir) and 'r' or 'l'
6729     end

```

```

6730     -- when the callback is called, we are just _after_ the box,
6731     -- and the texdir is that of the surrounding text
6732     -- if not ispar and hdir ~= tex.texdir then
6733     --   save_outer = ('TRT' == hdir) and 'r' or 'l'
6734     -- end
6735     local outer = save_outer
6736     local last = outer
6737     -- 'al' is only taken into account in the first, current loop
6738     if save_outer == 'al' then save_outer = 'r' end
6739
6740     local fontmap = Babel.fontmap
6741
6742     for item in node.traverse(head) do
6743
6744         -- In what follows, #node is the last (previous) node, because the
6745         -- current one is not added until we start processing the neutrals.
6746
6747         -- three cases: glyph, dir, otherwise
6748         if item.id == GLYPH
6749             or (item.id == 7 and item.subtype == 2) then
6750
6751             local d_font = nil
6752             local item_r
6753             if item.id == 7 and item.subtype == 2 then
6754                 item_r = item.replace -- automatic discs have just 1 glyph
6755             else
6756                 item_r = item
6757             end
6758             local chardata = characters[item_r.char]
6759             d = chardata and chardata.d or nil
6760             if not d or d == 'nsm' then
6761                 for nn, et in ipairs(ranges) do
6762                     if item_r.char < et[1] then
6763                         break
6764                     elseif item_r.char <= et[2] then
6765                         if not d then d = et[3]
6766                         elseif d == 'nsm' then d_font = et[3]
6767                         end
6768                     break
6769                 end
6770             end
6771             end
6772             d = d or 'l'
6773
6774             -- A short 'pause' in bidi for mapfont
6775             d_font = d_font or d
6776             d_font = (d_font == 'l' and 0) or
6777                 (d_font == 'nsm' and 0) or
6778                 (d_font == 'r' and 1) or
6779                 (d_font == 'al' and 2) or
6780                 (d_font == 'an' and 2) or nil
6781             if d_font and fontmap[d_font][item_r.font] then
6782                 item_r.font = fontmap[d_font][item_r.font]
6783             end
6784
6785             if new_d then
6786                 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
6787                 if inmath then
6788                     attr_d = 0

```

```

6789         else
6790             attr_d = node.get_attribute(item, ATDIR)
6791             attr_d = attr_d % 3
6792         end
6793         if attr_d == 1 then
6794             outer_first = 'r'
6795             last = 'r'
6796         elseif attr_d == 2 then
6797             outer_first = 'r'
6798             last = 'al'
6799         else
6800             outer_first = 'l'
6801             last = 'l'
6802         end
6803         outer = last
6804         has_en = false
6805         first_et = nil
6806         new_d = false
6807     end
6808
6809     if glue_d then
6810         if (d == 'l' and 'l' or 'r') ~= glue_d then
6811             table.insert(nodes, {glue_i, 'on', nil})
6812         end
6813         glue_d = nil
6814         glue_i = nil
6815     end
6816
6817     elseif item.id == DIR then
6818         d = nil
6819         new_d = true
6820
6821     elseif item.id == node.id'glue' and item.subtype == 13 then
6822         glue_d = d
6823         glue_i = item
6824         d = nil
6825
6826     elseif item.id == node.id'math' then
6827         inmath = (item.subtype == 0)
6828
6829     else
6830         d = nil
6831     end
6832
6833     -- AL <= EN/ET/ES      -- W2 + W3 + W6
6834     if last == 'al' and d == 'en' then
6835         d = 'an'          -- W3
6836     elseif last == 'al' and (d == 'et' or d == 'es') then
6837         d = 'on'          -- W6
6838     end
6839
6840     -- EN + CS/ES + EN      -- W4
6841     if d == 'en' and #nodes >= 2 then
6842         if (nodes[#nodes][2] == 'es' or nodes[#nodes][2] == 'cs')
6843             and nodes[#nodes-1][2] == 'en' then
6844             nodes[#nodes][2] = 'en'
6845         end
6846     end
6847

```



```

6848 -- AN + CS + AN          -- W4 too, because uax9 mixes both cases
6849 if d == 'an' and #nodes >= 2 then
6850   if (nodes[#nodes][2] == 'cs')
6851     and nodes[#nodes-1][2] == 'an' then
6852     nodes[#nodes][2] = 'an'
6853   end
6854 end
6855
6856 -- ET/EN                  -- W5 + W7->l / W6->on
6857 if d == 'et' then
6858   first_et = first_et or (#nodes + 1)
6859 elseif d == 'en' then
6860   has_en = true
6861   first_et = first_et or (#nodes + 1)
6862 elseif first_et then      -- d may be nil here !
6863   if has_en then
6864     if last == 'l' then
6865       temp = 'l'      -- W7
6866     else
6867       temp = 'en'    -- W5
6868     end
6869   else
6870     temp = 'on'      -- W6
6871   end
6872   for e = first_et, #nodes do
6873     if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
6874   end
6875   first_et = nil
6876   has_en = false
6877 end
6878
6879 -- Force mathdir in math if ON (currently works as expected only
6880 -- with 'l')
6881 if inmath and d == 'on' then
6882   d = ('TRT' == tex.mathdir) and 'r' or 'l'
6883 end
6884
6885 if d then
6886   if d == 'al' then
6887     d = 'r'
6888     last = 'al'
6889   elseif d == 'l' or d == 'r' then
6890     last = d
6891   end
6892   prev_d = d
6893   table.insert(nodes, {item, d, outer_first})
6894 end
6895
6896 outer_first = nil
6897
6898 end
6899
6900 -- TODO -- repeated here in case EN/ET is the last node. Find a
6901 -- better way of doing things:
6902 if first_et then      -- dir may be nil here !
6903   if has_en then
6904     if last == 'l' then
6905       temp = 'l'      -- W7
6906     else

```

```

6907         temp = 'en'    -- W5
6908     end
6909     else
6910         temp = 'on'      -- W6
6911     end
6912     for e = first_et, #nodes do
6913         if nodes[e][1].id == GLYPH then nodes[e][2] = temp end
6914     end
6915 end
6916
6917 -- dummy node, to close things
6918 table.insert(nodes, {nil, (outer == 'l') and 'l' or 'r', nil})
6919
6920 ----- NEUTRAL -----
6921
6922 outer = save_outer
6923 last = outer
6924
6925 local first_on = nil
6926
6927 for q = 1, #nodes do
6928     local item
6929
6930     local outer_first = nodes[q][3]
6931     outer = outer_first or outer
6932     last = outer_first or last
6933
6934     local d = nodes[q][2]
6935     if d == 'an' or d == 'en' then d = 'r' end
6936     if d == 'cs' or d == 'et' or d == 'es' then d = 'on' end --- W6
6937
6938     if d == 'on' then
6939         first_on = first_on or q
6940     elseif first_on then
6941         if last == d then
6942             temp = d
6943         else
6944             temp = outer
6945         end
6946         for r = first_on, q - 1 do
6947             nodes[r][2] = temp
6948             item = nodes[r][1]    -- MIRRORING
6949             if Babel.mirroring_enabled and item.id == GLYPH
6950                 and temp == 'r' and characters[item.char] then
6951                 local font_mode = font.fonts[item.font].properties.mode
6952                 if font_mode ~= 'harf' and font_mode ~= 'plug' then
6953                     item.char = characters[item.char].m or item.char
6954                 end
6955             end
6956         end
6957         first_on = nil
6958     end
6959
6960     if d == 'r' or d == 'l' then last = d end
6961 end
6962
6963 ----- IMPLICIT, REORDER -----
6964
6965 outer = save_outer

```

```

6966 last = outer
6967
6968 local state = {}
6969 state.has_r = false
6970
6971 for q = 1, #nodes do
6972
6973     local item = nodes[q][1]
6974
6975     outer = nodes[q][3] or outer
6976
6977     local d = nodes[q][2]
6978
6979     if d == 'nsm' then d = last end          -- W1
6980     if d == 'en' then d = 'an' end
6981     local isdir = (d == 'r' or d == 'l')
6982
6983     if outer == 'l' and d == 'an' then
6984         state.san = state.san or item
6985         state.ean = item
6986     elseif state.san then
6987         head, state = insert_numeric(head, state)
6988     end
6989
6990     if outer == 'l' then
6991         if d == 'an' or d == 'r' then      -- im -> implicit
6992             if d == 'r' then state.has_r = true end
6993             state.sim = state.sim or item
6994             state.eim = item
6995         elseif d == 'l' and state.sim and state.has_r then
6996             head, state = insert_implicit(head, state, outer)
6997         elseif d == 'l' then
6998             state.sim, state.eim, state.has_r = nil, nil, false
6999         end
7000     else
7001         if d == 'an' or d == 'l' then
7002             if nodes[q][3] then -- nil except after an explicit dir
7003                 state.sim = item -- so we move sim 'inside' the group
7004             else
7005                 state.sim = state.sim or item
7006             end
7007             state.eim = item
7008         elseif d == 'r' and state.sim then
7009             head, state = insert_implicit(head, state, outer)
7010         elseif d == 'r' then
7011             state.sim, state.eim = nil, nil
7012         end
7013     end
7014
7015     if isdir then
7016         last = d          -- Don't search back - best save now
7017     elseif d == 'on' and state.san then
7018         state.san = state.san or item
7019         state.ean = item
7020     end
7021
7022 end
7023
7024 return node.prev(head) or head

```

```
7025 end
7026 </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.

```
7027 <*nil>
7028 \ProvidesLanguage{nil}[<<date>>] <<version>> Nil language]
7029 \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.

```
7030 \ifx\l@nil\undefined
7031 \newlanguage\l@nil
7032 \@namedef{bbl@hyphendata@the\l@nil}{\relax}% Remove warning
7033 \let\bbl@elt\relax
7034 \edef\bbl@languages{% Add it to the list of languages
7035 \bbl@languages\bbl@elt{nil}{the\l@nil}}
7036 \fi
```

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

```
7037 \providehyphenmins{\CurrentOption}{\m@ne\m@ne}
```

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

```
\captionnil
\datenil
7038 \let\captionnil\empty
7039 \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.

```
7040 \ldf@finish{nil}
7041 </nil>
```



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).

```

7061 <<{*Emulate LaTeX}>> ≡
7062 \def\@empty{}
7063 \def\loadlocalcfg#1{%
7064   \openin0#1.cfg
7065   \ifeof0
7066     \closein0
7067   \else
7068     \closein0
7069     {\immediate\write16{*****}%
7070      \immediate\write16{* Local config file #1.cfg used}%
7071      \immediate\write16{*}%
7072     }
7073     \input #1.cfg\relax
7074   \fi
7075   \@endoflfd}

```

## 16.3 General tools

A number of  $\text{\LaTeX}$  macro's that are needed later on.

```

7076 \long\def\@firstofone#1{#1}
7077 \long\def\@firstoftwo#1#2{#1}
7078 \long\def\@secondoftwo#1#2{#2}
7079 \def\@nnil{\@nil}
7080 \def\@gobbletwo#1#2{}
7081 \def\@ifstar#1{\@ifnextchar *{\@firstoftwo{#1}}}
7082 \def\@star@or@long#1{%
7083   \@ifstar
7084   {\let\l@ngrel@x\relax#1}%
7085   {\let\l@ngrel@x\long#1}}
7086 \let\l@ngrel@x\relax
7087 \def\@car#1#2\@nil{#1}
7088 \def\@cdr#1#2\@nil{#2}
7089 \let\@typeset@protect\relax
7090 \let\protected@edef\edef
7091 \long\def\@gobble#1{}
7092 \edef\@backslashchar{\expandafter\@gobble\string\}
7093 \def\strip@prefix#1>{}
7094 \def\g@addto@macro#1#2{%
7095   \toks@\expandafter{#1#2}%
7096   \xdef#1{\the\toks@}}
7097 \def\@namedef#1{\expandafter\def\csname #1\endcsname}
7098 \def\@nameuse#1{\csname #1\endcsname}
7099 \def\@ifundefined#1{%
7100   \expandafter\ifx\csname#1\endcsname\relax
7101     \expandafter\@firstoftwo
7102   \else
7103     \expandafter\@secondoftwo
7104   \fi}
7105 \def\@expandtwoargs#1#2#3{%
7106   \edef\reserved@a{\noexpand#1{#2}{#3}}\reserved@a}
7107 \def\zap@space#1 #2{%
7108   #1%
7109   \ifx#2\@empty\else\expandafter\zap@space\fi
7110   #2}
7111 \let\bbl@trace\@gobble
7112 \def\bbl@error#1#2{%

```

```

7113 \begingroup
7114   \newlinechar=`^^J
7115   \def\{^^J(babel) }%
7116   \errhelp{#2}\errmessage{\#1}%
7117 \endgroup}
7118 \def\bbl@warning#1{%
7119   \begingroup
7120     \newlinechar=`^^J
7121     \def\{^^J(babel) }%
7122     \message{\#1}%
7123   \endgroup}
7124 \let\bbl@infowarn\bbl@warning
7125 \def\bbl@info#1{%
7126   \begingroup
7127     \newlinechar=`^^J
7128     \def\{^^J}%
7129     \wlog{#1}%
7130   \endgroup}

```

$\text{\LaTeX}$  2<sub>ε</sub> has the command `\@onlypreamble` which adds commands to a list of commands that are no longer needed after `\begin{document}`.

```

7131 \ifx\@preamblecmds\@undefined
7132   \def\@preamblecmds{}
7133 \fi
7134 \def\@onlypreamble#1{%
7135   \expandafter\gdef\expandafter\@preamblecmds\expandafter{%
7136     \@preamblecmds\do#1}}
7137 \@onlypreamble\@onlypreamble

```

Mimick  $\text{\LaTeX}$ 's `\AtBeginDocument`; for this to work the user needs to add `\begindocument` to his file.

```

7138 \def\begindocument{%
7139   \@begindocumenthook
7140   \global\let\@begindocumenthook\@undefined
7141   \def\do##1{\global\let##1\@undefined}%
7142   \@preamblecmds
7143   \global\let\do\noexpand}
7144 \ifx\@begindocumenthook\@undefined
7145   \def\@begindocumenthook{}
7146 \fi
7147 \@onlypreamble\@begindocumenthook
7148 \def\AtBeginDocument{\g@addto@macro\@begindocumenthook}

```

We also have to mimick  $\text{\LaTeX}$ 's `\AtEndOfPackage`. Our replacement macro is much simpler; it stores its argument in `\@endofldf`.

```

7149 \def\AtEndOfPackage#1{\g@addto@macro\@endofldf{#1}}
7150 \@onlypreamble\AtEndOfPackage
7151 \def\@endofldf{}
7152 \@onlypreamble\@endofldf
7153 \let\bbl@afterlang\@empty
7154 \chardef\bbl@opt@hyphenmap\z@

```

$\text{\LaTeX}$  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.

```

7155 \catcode`\&=\z@
7156 \ifx&\if@files\@undefined
7157   \expandafter\let\csname if@files\expandafter\endcsname
7158     \csname iffalse\endcsname
7159 \fi
7160 \catcode`\&=4

```

Mimick L<sup>A</sup>T<sub>E</sub>X's commands to define control sequences.

```

7161 \def\newcommand{\@star@or@long\new@command}
7162 \def\new@command#1{%
7163   \@testopt{\@newcommand#1}0}
7164 \def\@newcommand#1[#2]{%
7165   \@ifnextchar [{\@xargdef#1[#2]]%
7166     {\@argdef#1[#2]}}
7167 \long\def\@argdef#1[#2]#3{%
7168   \@yargdef#1\@ne{#2}{#3}}
7169 \long\def\@xargdef#1[#2][#3]#4{%
7170   \expandafter\def\expandafter#1\expandafter{%
7171     \expandafter\@protected@testopt\expandafter #1%
7172     \csname\string#1\expandafter\endcsname{#3}}%
7173   \expandafter\@yargdef \csname\string#1\endcsname
7174   \tw@{#2}{#4}}
7175 \long\def\@yargdef#1#2#3{%
7176   \@tempcnta#3\relax
7177   \advance \@tempcnta \@ne
7178   \let\@hash@\relax
7179   \edef\reserved@a{\ifx#2\tw@ [\@hash@1]\fi}%
7180   \@tempcntb #2%
7181   \@whilenum \@tempcntb <\@tempcnta
7182   \do{%
7183     \edef\reserved@a{\reserved@a\@hash@the\@tempcntb}%
7184     \advance\@tempcntb \@ne}%
7185   \let\@hash@###
7186   \l@ngrelx\expandafter\def\expandafter#1\reserved@a}
7187 \def\providecommand{\@star@or@long\provide@command}
7188 \def\provide@command#1{%
7189   \begingroup
7190     \escapechar\m@ne\xdef\@gtempa{{\string#1}}%
7191   \endgroup
7192   \expandafter\ifundefined\@gtempa
7193     {\def\reserved@a{\new@command#1}}%
7194     {\let\reserved@a\relax
7195      \def\reserved@a{\new@command\reserved@a}}%
7196   \reserved@a}%
7197 \def\DeclareRobustCommand{\@star@or@long\declare@robustcommand}
7198 \def\declare@robustcommand#1{%
7199   \edef\reserved@a{\string#1}%
7200   \def\reserved@b{#1}%
7201   \edef\reserved@b{\expandafter\strip@prefix\meaning\reserved@b}%
7202   \edef#1{%
7203     \ifx\reserved@a\reserved@b
7204       \noexpand\x@protect
7205       \noexpand#1%
7206     \fi
7207     \noexpand\protect
7208     \expandafter\noexpand\csname
7209       \expandafter\@gobble\string#1 \endcsname
7210   }%
7211   \expandafter\new@command\csname
7212     \expandafter\@gobble\string#1 \endcsname
7213 }
7214 \def\x@protect#1{%
7215   \ifx\protect\@typeset@protect\else
7216     \@x@protect#1%
7217   \fi

```



```

7218 }
7219 \catcode`\&=\z@ % Trick to hide conditionals
7220 \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`.

```

7221 \def\bbl@tempa{\csname newif\endcsname&fin@}
7222 \catcode`\&=4
7223 \ifx\in@\@undefined
7224 \def\in@#1#2{%
7225 \def\in@##1#1##2##3\in@{%
7226 \ifx\in@##2\in@false\else\in@true\fi}%
7227 \in@##2#1\in@\in@}
7228 \else
7229 \let\bbl@tempa\@empty
7230 \fi
7231 \bbl@tempa

```

$\text{\LaTeX}$  has a macro to check whether a certain package was loaded with specific options. The command has two extra arguments which are code to be executed in either the true or false case. This is used to detect whether the document needs one of the accents to be activated (activegrave and activeacute). For plain  $\text{\TeX}$  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).

```

7232 \def\@ifpackagewith#1#2#3#4{#3}

```

The  $\text{\LaTeX}$  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.

```

7233 \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  $\text{\LaTeX}_{2\epsilon}$  versions; just enough to make things work in plain  $\text{\TeX}$  environments.

```

7234 \ifx\@tempcnta\@undefined
7235 \csname newcount\endcsname\@tempcnta\relax
7236 \fi
7237 \ifx\@tempcntb\@undefined
7238 \csname newcount\endcsname\@tempcntb\relax
7239 \fi

```

To prevent wasting two counters in  $\text{\LaTeX}$  (because counters with the same name are allocated later by it) we reset the counter that holds the next free counter (`\count10`).

```

7240 \ifx\bye\@undefined
7241 \advance\count10 by -2\relax
7242 \fi
7243 \ifx\@ifnextchar\@undefined
7244 \def\@ifnextchar#1#2#3{%
7245 \let\reserved@d=#1%
7246 \def\reserved@a{#2}\def\reserved@b{#3}%
7247 \futurelet\@let@token\@ifnch}
7248 \def\@ifnch{%
7249 \ifx\@let@token\sptoken
7250 \let\reserved@c\@ifnch
7251 \else
7252 \ifx\@let@token\reserved@d
7253 \let\reserved@c\reserved@a
7254 \else
7255 \let\reserved@c\reserved@b
7256 \fi
7257 \fi

```

```

7258 \reserved@c}
7259 \def\:{\let\sptoken= } \: % this makes \@sptoken a space token
7260 \def\:{\@xifnch} \expandafter\def\:{\futurelet\@let@token\@ifnch}
7261 \fi
7262 \def\@testopt#1#2{%
7263 \ifnextchar[{\#1}{\#1[#2]}}
7264 \def\@protected@testopt#1{%
7265 \ifx\protect\@typeset@protect
7266 \expandafter\@testopt
7267 \else
7268 \@x@protect#1%
7269 \fi}
7270 \long\def\@whilenum#1\do #2{\ifnum #1\relax #2\relax\@iwhilenum{#1\relax
7271 #2\relax}\fi}
7272 \long\def\@iwhilenum#1{\ifnum #1\expandafter\@iwhilenum
7273 \else\expandafter\@gobble\fi{#1}}

```

## 16.4 Encoding related macros

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

```

7274 \def\DeclareTextCommand{%
7275 \@dec@text@cmd\providecommand
7276 }
7277 \def\ProvideTextCommand{%
7278 \@dec@text@cmd\providecommand
7279 }
7280 \def\DeclareTextSymbol#1#2#3{%
7281 \@dec@text@cmd\chardef#1{#2}#3\relax
7282 }
7283 \def\@dec@text@cmd#1#2#3{%
7284 \expandafter\def\expandafter#2%
7285 \expandafter{%
7286 \csname#3-cmd\expandafter\endcsname
7287 \expandafter#2%
7288 \csname#3\string#2\endcsname
7289 }%
7290 % \let\@ifdefinable\rc@ifdefinable
7291 \expandafter#1\csname#3\string#2\endcsname
7292 }
7293 \def\@current@cmd#1{%
7294 \ifx\protect\@typeset@protect\else
7295 \noexpand#1\expandafter\@gobble
7296 \fi
7297 }
7298 \def\@changed@cmd#1#2{%
7299 \ifx\protect\@typeset@protect
7300 \expandafter\ifx\csname\cf@encoding\string#1\endcsname\relax
7301 \expandafter\ifx\csname ?\string#1\endcsname\relax
7302 \expandafter\def\csname ?\string#1\endcsname{%
7303 \@changed@x@err{#1}%
7304 }%
7305 \fi
7306 \global\expandafter\let
7307 \csname\cf@encoding\string#1\expandafter\endcsname
7308 \csname ?\string#1\endcsname
7309 \fi
7310 \csname\cf@encoding\string#1%
7311 \expandafter\endcsname

```

```

7312 \else
7313 \noexpand#1%
7314 \fi
7315 }
7316 \def\@changed@x@err#1{%
7317 \errhelp{Your command will be ignored, type <return> to proceed}%
7318 \errmessage{Command \protect#1 undefined in encoding \cf@encoding}}
7319 \def\DeclareTextCommandDefault#1{%
7320 \DeclareTextCommand#1?%
7321 }
7322 \def\ProvideTextCommandDefault#1{%
7323 \ProvideTextCommand#1?%
7324 }
7325 \expandafter\let\csname OT1-cmd\endcsname\@current@cmd
7326 \expandafter\let\csname?-cmd\endcsname\@changed@cmd
7327 \def\DeclareTextAccent#1#2#3{%
7328 \DeclareTextCommand#1{#2}[1]{\accent#3 ##1}
7329 }
7330 \def\DeclareTextCompositeCommand#1#2#3#4{%
7331 \expandafter\let\expandafter\reserved@a\csname#2\string#1\endcsname
7332 \edef\reserved@b{\string##1}%
7333 \edef\reserved@c{%
7334 \expandafter\@strip@args\meaning\reserved@a:-\@strip@args}%
7335 \ifx\reserved@b\reserved@c
7336 \expandafter\expandafter\expandafter\ifx
7337 \expandafter\@car\reserved@a\relax\relax\@nil
7338 \@text@composite
7339 \else
7340 \edef\reserved@b##1{%
7341 \def\expandafter\noexpand
7342 \csname#2\string#1\endcsname####1{%
7343 \noexpand\@text@composite
7344 \expandafter\noexpand\csname#2\string#1\endcsname
7345 ####1\noexpand\@empty\noexpand\@text@composite
7346 {##1}%
7347 }%
7348 }%
7349 \expandafter\reserved@b\expandafter{\reserved@a{##1}}%
7350 \fi
7351 \expandafter\def\csname\expandafter\string\csname
7352 #2\endcsname\string#1-\string#3\endcsname{#4}
7353 \else
7354 \errhelp{Your command will be ignored, type <return> to proceed}%
7355 \errmessage{\string\DeclareTextCompositeCommand\space used on
7356 inappropriate command \protect#1}
7357 \fi
7358 }
7359 \def\@text@composite#1#2#3\@text@composite{%
7360 \expandafter\@text@composite@x
7361 \csname\string#1-\string#2\endcsname
7362 }
7363 \def\@text@composite@x#1#2{%
7364 \ifx#1\relax
7365 #2%
7366 \else
7367 #1%
7368 \fi
7369 }
7370 %

```

```

7371 \def\@strip@args#1:#2-#3\@strip@args{#2}
7372 \def\DeclareTextComposite#1#2#3#4{%
7373   \def\reserved@a{\DeclareTextCompositeCommand#1{#2}{#3}}%
7374   \bgroup
7375     \lcode` \@=#4%
7376     \lowercase{%
7377   \egroup
7378     \reserved@a @%
7379   }%
7380 }
7381 %
7382 \def\UseTextSymbol#1#2{#2}
7383 \def\UseTextAccent#1#2#3{}
7384 \def\@use@text@encoding#1{}
7385 \def\DeclareTextSymbolDefault#1#2{%
7386   \DeclareTextCommandDefault#1{\UseTextSymbol{#2}#1}%
7387 }
7388 \def\DeclareTextAccentDefault#1#2{%
7389   \DeclareTextCommandDefault#1{\UseTextAccent{#2}#1}%
7390 }
7391 \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.

```

7392 \DeclareTextAccent{"}{OT1}{127}
7393 \DeclareTextAccent{'}{OT1}{19}
7394 \DeclareTextAccent{^}{OT1}{94}
7395 \DeclareTextAccent`{OT1}{18}
7396 \DeclareTextAccent{~}{OT1}{126}

```

The following control sequences are used in `babel.def` but are not defined for `PLAIN  $\text{\TeX}$` .

```

7397 \DeclareTextSymbol{\textquotedblleft}{OT1}{92}
7398 \DeclareTextSymbol{\textquotedblright}{OT1}{`"}
7399 \DeclareTextSymbol{\textquoteleft}{OT1}{``}
7400 \DeclareTextSymbol{\textquoteright}{OT1}{''}
7401 \DeclareTextSymbol{\i}{OT1}{16}
7402 \DeclareTextSymbol{\ss}{OT1}{25}

```

For a couple of languages we need the  $\text{\LaTeX}$ -control sequence `\scriptsize` to be available. Because plain  $\text{\TeX}$  doesn't have such a sophisticated font mechanism as  $\text{\LaTeX}$  has, we just `\let` it to `\sevenrm`.

```

7403 \ifx\scriptsize\@undefined
7404   \let\scriptsize\sevenrm
7405 \fi

```

And a few more “dummy” definitions.

```

7406 \def\language{english}%
7407 \let\bbl@opt@shorthands\@nnil
7408 \def\bbl@ifshorthand#1#2#3{#2}%
7409 \let\bbl@language@opts\@empty
7410 \ifx\babeloptionstrings\@undefined
7411   \let\bbl@opt@strings\@nnil
7412 \else
7413   \let\bbl@opt@strings\babeloptionstrings
7414 \fi
7415 \def\BabelStringsDefault{generic}
7416 \def\bbl@tempa{normal}
7417 \ifx\babeloptionmath\bbl@tempa
7418   \def\bbl@mathnormal{\noexpand\textormath}
7419 \fi

```

```

7420 \def\AfterBabelLanguage#1#2{}
7421 \ifx\BabelModifiers\undefined\let\BabelModifiers\relax\fi
7422 \let\bbl@afterlang\relax
7423 \def\bbl@opt@safe{BR}
7424 \ifx\uclclist\undefined\let\uclclist\empty\fi
7425 \ifx\bbl@trace\undefined\def\bbl@trace#1{}\fi
7426 \expandafter\newif\csname ifbbl@single\endcsname
7427 \chardef\bbl@bidimode\z@
7428 <</Emulate LaTeX>>

```

A proxy file:

```

7429 <*plain>
7430 \input babel.def
7431 </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  $\LaTeX$  styles*, *TUGboat* 10 (1989) #3, p. 401–406.
- [3] Yannis Haralambous, *Fonts & Encodings*, O'Reilly, 2007.
- [4] Donald E. Knuth, *The  $\TeX$ book*, Addison-Wesley, 1986.
- [5] Jukka K. Korpela, *Unicode Explained*, O'Reilly, 2006.
- [6] Leslie Lamport,  *$\LaTeX$ , A document preparation System*, Addison-Wesley, 1986.
- [7] Leslie Lamport, in:  $\TeX$ hax Digest, Volume 89, #13, 17 February 1989.
- [8] Ken Lunde, *CJKV Information Processing*, O'Reilly, 2nd ed., 2009.
- [9] Hubert Partl, *German  $\TeX$* , *TUGboat* 9 (1988) #1, p. 70–72.
- [10] Joachim Schrod, *International  $\LaTeX$  is ready to use*, *TUGboat* 11 (1990) #1, p. 87–90.
- [11] Apostolos Syropoulos, Antonis Tsolomitis and Nick Sofroniu, *Digital typography using  $\LaTeX$* , Springer, 2002, p. 301–373.
- [12] K.F. Treebus. *Tekstwijzer, een gids voor het grafisch verwerken van tekst*, SDU Uitgeverij ('s-Gravenhage, 1988).